ENTERPRISE IDENTIFIERS
FOR LOGISTICS

An Approach in Support of Army Transformation Initiatives

Prepared For:

Mr. Bruce Haberkamp
ATTN: SAIS-PAA-S
Room 1C634, 107 Army Pentagon
Washington, DC 20310-0107

Prepared By:

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5067

and

Amerind
Delivering Solutions That Delight Customers

1310 Braddock Place
Suite 400
Alexandria, Virginia 22314-1648

and

IDA

Institute for Defense Analyses
1801 N. Beauregard St.
Alexandria, VA 22311

PREFACE

This study resulted from a request from Brigadier General H.A. Curry (Director for Plans, Operations, and Logistics Automation in the ODCSLOG) pursuant to a briefing given to him on 6 July 1999. The notion of using "organization" as the unifying concept between Operations and Logistics was introduced at this briefing. This paper documents the recommendations of the project for expanding to the area of Logistics the concept of globally unique 64-bit identifiers (enterprise identifiers) that are built from a 32-bit identifier moiety called the ORGANIZATION-IDENTIFIER. The Army Studies Program provided the funding for this study.

.

ACKNOWLEDGEMENTS

We wish to acknowledge the assistance and contributions of the following individuals who responded to our requests for information on industry experience with the use of surrogate and enterprise keys:

- **Dr. Tom Johnston** – an independent consultant specializing in relational and object-oriented data architecture and data modeling. Dr. Johnston was the author of the six articles in *Data Management Review* on the topic of enterprise keys that provided valuable insight and validation for the project team during the development of this proposal. In addition, his observations, comments, and suggestions during the project were very helpful to the team's deliberations concerning issues and alternative solutions for implementation.

- **Mr. Frank Palmeri** – a data administrator with the New York State Tax and Finance Department. Mr. Palmeri volunteered information on a presentation concerning the JB Hunt Transport Company's project that implemented surrogate enterprise keys into their information system databases. In addition, his comments concerning the apparent lack of understanding of the concept by relational modelers at the conference reinforced the view that pure relational modelers are not as conversant with the need for globally unique identifiers as are object-oriented modelers.

- **Dr. Ralph Kimball** – a nationally recognized expert database designer and author of numerous books on designing data warehouse database systems. Dr. Kimball graciously permitted us to quote him directly on the need for using surrogate identifiers as the primary keys of relational databases.

- **Ms. Terry Moriarty** – a nationally prominent database designer and data modeler. She is President of Inastrol, an information management consulting firm, and a frequent lecturer at professional forums on data management topics. Ms. Moriarty responded to our request for information on the use of surrogate primary keys. Her insights and opinions on the superiority of surrogate identifiers as primary keys were greatly appreciated and provided valuable validation for the concept of enterprise keys.

ENTERPRISE IDENTIFIERS FOR LOGISTICS
An Approach in Support of Army Transformation Initiatives

Table of Contents

APPENDICES

LIST OF FIGURES

EXECUTIVE SUMMARY

Introduction

Currently the main thrust of the U.S. Army is the implementation of Joint Vision 2010. To accomplish this, the Army is engaged in a large scale "transformation" of its forces. Vital to this transformation is the efficient and reliable management of its information.

To quote the Chief of Staff for the Army, General Eric K. Shinseki, "Our commitment to meeting these challenges compels comprehensive transformation of *The Army*. To this end, we will begin immediately to transition the entire Army into a force that is strategically responsive and dominant at every point on the spectrum of operations. We will jumpstart the process by investing in today's off-the-shelf technology to stimulate the development of doctrine, organizational design, and leader training even as we begin a search for new technologies for the objective force. Doing so will extend our technological overmatch."

Thus an essential part of this vision is to develop the capability to rapidly designate, marshal, deploy, and employ joint, tailored military forces designed to meet specific needs and requirements in response to a wide range of possible worldwide contingencies. The capability must also exist for field commanders to rapidly restructure the forces under their command to respond to changing battlefield situations. This requires a capability of accessing the three principal force package data objects (the organization structure, personnel, and materiel) with confidence in their accuracy and reliability. In spite of the large number of information systems, the current situation is such that commanders are not able to obtain timely delivery of these capabilities. Untimely delivery of critical data is largely the result of the following deficiencies: (1) lack of a single, universal approach for information identification management, and (2) lack of unambiguous sources for standard data.

Study Purpose

This study began as a result of a briefing to Brigadier General H.A. Curry, (Director of Plans, Operations, and Logistics Information Management, Headquarters, Department of the Army (HQDA), Office of the Deputy Chief of Staff for Logistics (ODCSLOG) that took place on July 6, 1999. During this briefing, the concept was introduced of using globally unique identifiers or *enterprise identifiers (EIDs)* based on "organization" as a mechanism for unifying data across functions such as Operations and Logistics. This briefing was based on the work being done on the Army Battle Command System (ABCS). As a result of the presentation, General Curry asked these two questions:

1. Can the Logistics community exploit a common, detailed organizational structure?

2. Can the approach be realistically implemented with legacy systems?

This study answers these two questions and identifies two specific areas where this concept can be exploited:

1. Building Units on the Fly (BUOF)[1], and

2. Integrating Logistics and Command & Control information systems.

Problem Statement

BUOF is a requirement for building a force and generating the Level 6 Time-Phased Force and Deployment Data (TPFDD) Plan in less than 72 hours. Due to the stovepipe nature of the legacy systems environment, BUOF is a manually labor intensive task that presently cannot be achieved in a timely manner.

In addition, battlefield commanders and even logisticians cannot easily make use of Logistics data to manage their resources because there is no standard way of identifying and locating Logistics data records. This is because:

- Static and stationary[2] Logistics data is fragmented, and unsynchronized both across and within battle command and support systems. Similarly, in tactical operations, commanders cannot exchange dynamic data easily between systems. This symptom is true of both Army and joint systems.

- The supply chain is comprised of many different systems that do not employ a consistent method of identifying and locating Logistics data.

Proposed Solution

1. Adopt an identifying convention or enterprise identifier (EID) scheme, for locating and managing assets in support of the warfighter throughout the entire spectrum of combat operations. The EID scheme will allow objects across the combat, combat support, and combat service support domains to be uniquely identified.

2. Establish and maintain an authoritative source of stationary Logistics data using a common EID scheme to ensure that data can be easily and reliably accessed with confidence in its accuracy. From a system management perspective, defining the authoritative source will make clear which data systems are the definitive source or information systems of record for various aspects of Army Logistics equipment and materiel data. Organizations should also be identified to exercise management authority over the related databases to control access for creating, deleting, and editing records.

---

1 Contract Number DASW01-98-D-0060. Briefing by Col. (Ret) D. Measels, SAIC, 13 Oct 99.

2 Stationary data is information that is "semi-static," meaning that it is relatively invariant over its lifetime and can be maintained in a shared reference library. An example of semi-static data is phone numbers.

3. Develop a server for the Logistics data domain. Initially, a Materiel server should be developed for materiel-type-related data. The data furnished by the server would implement a common DoD data schema for data about types of materiel.

4. Base the EID scheme on the Army Organization Server (AOS) concept included in the operational requirements document (ORD) for the Force Management System (FMS) being implemented by U.S. Army Force Management Support Agency (USAFMSA) for managing military units. The AOS includes a unique integer identifier for any DoD organization based on the premise that "organization" is the central theme by which all battlefield objects relate (see Figure A). USAFMSA presently has a prototype AOS that contains selected Army forces down to the billet level. Ultimately, the goal is to produce a suite of servers that contain stationary data that is maintained by the agencies that have proponency for it. In other words, the servers are maintained by those who already have responsibility for the data and maintain it on a constant basis (see Figure B).

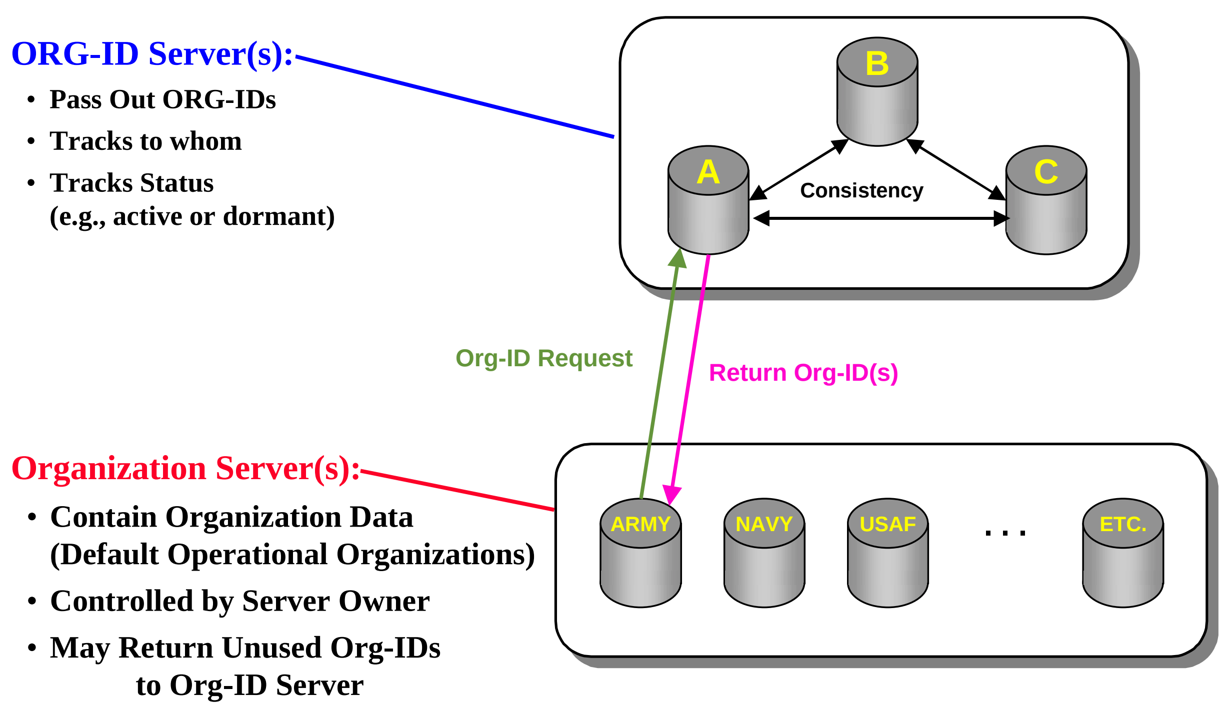


Figure A. ORG-ID Servers Provide ORG-IDs For Organizations That Maintain Force Structure Data

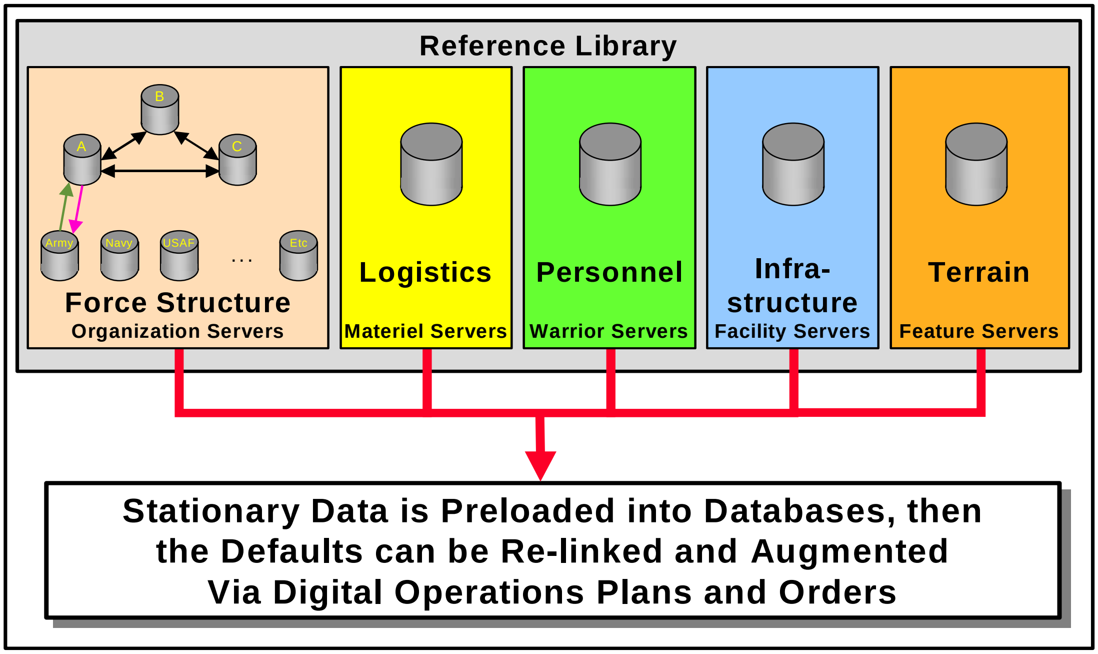


Figure B. Ultimate Goal, A Suite of Servers Containing Stationary Data

Recommended Actions

1. Designate an authoritative source within the Army for stationary Logistics data. (For example, LOGSA currently maintains a repository of Army Logistics data in the Logistics Integrated Database (LIDB)).

2. Populate a prototype Materiel server beginning with MATERIEL-TYPE catalog data.

3. Interface the prototype Materiel server proposed in (3) above, with current Logistics related warfighter systems (e.g. Standard Army Information System (STAMIS) and ABCS systems).

Conclusion

The EID concept provides an unprecedented opportunity to enable seamless interoperability of Army data systems. The EID concept provides a unifying theme (the force structure) and creates a single, globally recognized, unique identifier for all battlefield assets such as Materiel, Personnel, etc. This will permit Logistics systems to share all organizationally related Logistics data seamlessly and transparently with confidence in the consistency and integrity of the data. It will give Logistics planners and commanders the ability to quickly assemble ad-hoc force packages in support of the Army BUOF concept, and the Joint Staff initiative for developing a 72-hour TPFDD capability.

1.0 C4I-LOGISTICS OPERATIONAL CONCEPT

1.1 Introduction

Command, Control, Communications, Computer, Intelligence, Surveillance and Reconnaissance (C4ISR) are transforming along with the rest of the Army. The interoperability concept for C4ISR systems is to become "organization-centric[3] versus the current Logistics model which is "materiel-centric." This simple, but crucial difference is the basis for the problems encountered attempting to integrate Logistics into our Command & Control (C2) systems.

To share a common understanding of the battlespace required by the Objective Force, C4ISR and Logistics information systems must transform to the same organization-centric style. Three actions will enable us to accomplish this:

- Applying the military science thinking piece "up-front."
- Generating a default organization structure, that includes all functional requirements, for common use in both C2 and Logistics information systems.
- Synchronizing C2 and Logistics information systems to use the same data.

1.2 The Military Science Aspect

For the warfighter, organizations are the basis for aggregating functional requirements. As these requirements change, based on the mission needs, the task organization changes to fulfill those needs. The one constant is this *concept* of the organizational structure, regardless of the mission it is performing[4]. Because the organizational structure is a constant[5], it should be the basis for passing information around the battlespace.

In Logistics, the emphasis is on the "materiel." A battalion commander will want to know the operational status of Alpha Company (e.g., 13 of 14 tanks are operational). The focus of the Logistics system is the status of vehicle A-21. Currently, a commander normally goes to two different places to get this information (e.g. the S-3 will know the up-to-the-minute current operational status and the S-4 will know the supply/maintenance status for all *reported* non-operational items).

Operator requirements placed on the logistician ultimately cause this difference in views of the battlespace between C2 and Logistics. Operators are concerned with the command and control of units and, for the most part, do not track individual items unless it affects (or will effect) their

---

[3] Organization-centric implies that the military unit, or the force structure, is the basis for aggregating and integrating information exchange in the battlespace.

[4] For example: A tank platoon's primary mission is destroying tanks. However, it could receive a mission to fight forest fires. The concept of the "platoon" organization is a constant.

[5] It is a very rare occurrence where a "new" military organization is created "on the fly" (i.e. one does not purchase equipment and hire new people to operate that equipment.) Organization changes occur in MTOE documents. Task Organization is dynamic, but it is simply a rearrangement of equipment and people to perform a specific mission.

units' operational capabilities. They delegate the tracking of these items to the logistician, who have developed independent systems to handle these requirements.

What we are attempting to do in C4ISR is reduce the "Iron Mountain" [6] and the number of required logisticians by exploiting sound and timely information. The premise is that one can use information to replace supply stockpiles and maintainers. This is done by automating routine supply tasks, better informing logistical systems of actual unit requirements, and anticipating requirements. Fewer logisticians are required to process and track requests and, instead of sending a "standard" log pack, log packs are tailored to the actual unit needs. Also fewer soldiers are needed to process requests, fewer drivers need to move the needed items, and less storage space is required for highly mobile warfighting units. This is a "Win-Win" situation for the operators and logisticians.

To support the Army Transformation effort, we must integrate Logistics and C2. Not only does the operator require a unit's status, but also operators need to know the impact of Logistics on future operations. Logistics information exchanges must be fully integrated into the C2 network down to the tactical level. This can happen if we apply our doctrine now, at the start of the re-engineering process. What we need is a capability that provides Logistics information to the warfighter for use in C2 systems, while simultaneously fulfilling all logistical functions in support of the warfighter's needs. This is a military science problem long before it becomes a technological problem.

1.3 Generating a Default Organization Structure

To get the maximum benefits of digitization, we must pull our doctrine and Tactics, Techniques, and Procedures (TTPs) from the Field Manuals (FMs) and apply it to our force structure. If we define our default organization structure to reflect warfighter requirements at the start of the process, we've given all other systems a central concept to build to. The new Force Management System (FMS), currently under development by the U.S. Army Force Management Support Agency (USAFMSA), under the DCSOPS, is designed to add doctrine back into our force structure documentation. It transitions the MTOE[7] from a Logistics-based document (the soldier section and the equipment section) to an organizational-based, operational document (with people and equipment assigned to organizations). This makes MTOEs look more like an "organization chart", aiding our conceptual picture of the functions the organization is designed to perform. FMS also gives each organization a globally unique name, even in joint and multi-national environments. This name is known as the ORGANIZATION IDENTIFIER (ORG-ID).

1.4 Synchronization

ORG-ID (and the force structure that comes with it) allows the Logistics systems to logically interface with the C2 systems. The use of ORG-ID will allow Logistics applications to work seamlessly with C2 applications. This also facilitates the use of the same C2 networks (e.g. the

---

6 The current verbiage is that the "Distribution Based Logistics System (DBLS)" will replace the Iron Mountain and that the C4ISR will help get to the DBLS.

7 MTOE: Modified Table of Organization and Equipment.

Tactical Internet), the same communications backbone (e.g. WIN-T[8]) and will give Logistics systems at all echelons access to a global reach-back capability through the Global Information Grid (GIG). ORG-IDs are the simplest of the standards - a common naming convention for organizations. The power of this is using the ORG-ID as the basis for uniquely naming all of the other system data generated. ORG-IDs may be used to create Enterprise Identifiers (EIDs), which provide a flexible approach for accomplishing a common naming convention for all data.

1.5 Summary

For the operator and the logisticians, the payoffs are huge. Consider a requirement to build a force package capable of rapidly deploying.

- Using FMS, you design the force structure, based on desired functionality.
- With this new ORGANIZATION also come the support requirements. If you choose a Tank company, FMS would be able to tell you what the direct and general support maintenance requirements are for this force structure.
- Built into the force structure are the transportation requirements. It details what it will take to move the deployed force, in terms of ships, sorties, or railcars.
- The next step would be to select the actual unit and put the plan into motion.

To better support the operator, there is a need for tying the Logistics information to the force structure. Likewise, integrating Logistics to the C2 systems gives logisticians access to all of the C2 information, as well as their communications networks. To share a common understanding of the battlespace required by the Objective Force, C4ISR and Logistics information systems must transform to the same organization-centric style. Three actions will enable us to accomplish this:

- Applying the military science thinking piece “up-front.”
- Generating a default organization structure, that includes all functional requirements, for common use in both C2 and Logistics information systems.
- Synchronizing C2 and Logistics information systems to use the same data.

[8] WIN-T: Warfighter Information Network – Tactical.

2.0 OPERATIONAL DETAIL

2.1 Introduction

Currently the main thrust of the U.S. Army is the implementation of Joint Vision 2010. To accomplish this, the Army is engaged in a large scale "transformation" of its forces. Vital to this transformation is the efficient and reliable management of its information.

To quote the Chief of Staff for the Army, General Eric K. Shinseki, "Our commitment to meeting these challenges compels comprehensive transformation of *The Army*. To this end, we will begin immediately to transition the entire Army into a force that is strategically responsive and dominant at every point on the spectrum of operations. We will jumpstart the process by investing in today's off-the-shelf technology to stimulate the development of doctrine, organizational design, and leader training even as we begin a search for new technologies for the objective force. Doing so will extend our technological overmatch."

Thus an essential part of this vision is to develop the capability to rapidly designate, marshal, deploy, and employ joint, tailored military forces designed to meet specific needs and requirements in response to a wide range of possible worldwide contingencies. The capability must also exist for field commanders to rapidly restructure the forces under their command to respond to changing battlefield situations. This requires a capability of accessing the three principal force package data objects (the organization structure, personnel, and materiel) with confidence in their accuracy and reliability. In spite of the large number of information systems, the current situation is such that commanders are not able to obtain timely delivery of these capabilities. Untimely delivery of critical data is largely the result of the following deficiencies: (1) lack of a single, universal approach for information identification management, and (2) lack of unambiguous sources for standard data.

2.1.1 Categorization of Data According to Persistence

In addressing deficiency (2) "lack of unambiguous sources for standard data" mentioned in the previous section, we soon discover that the requirements and management of data vary substantially, depending on how quickly it changes. Thus, "static data", i.e., data that changes only very slowly, such as the U.S. state codes, can be pre-loaded once and reused many times. At the other end of the spectrum, "dynamic data", e.g., sensor data streams, requires optimized 'real-time' or 'near real-time' information systems to handle it, and may not be subject to exchange in its raw form. For example, among C2 systems only post-processed sensor information, i.e., confirmed 'tracks' built out of many radar returns, are exchanged. There is, however, a third category of data, which changes faster than "static data" but much slower than true "dynamic data". For lack of a better term, this type of data will be referred to as "stationary data" in this paper. An example of "stationary data" is phone numbers. Although thousands of phone numbers are added, deleted, and changed daily worldwide, it is not a frequent event for a person's phone number to change. Usually, a person's phone number is static for the duration of their tour of duty. Another example of "stationary data" is the official listings of items that can be procured through DLA.

A third, and perhaps even more relevant example of "stationary data," is data related to the C2 data schemas pertaining to the five basic battlefield entities: ORGANIZATION, MATERIEL, PERSONNEL, FACILITY, and FEATURE. These five domains contain large amounts of information that can be considered "stationary data". This means that the data can reside in common reference libraries (e.g., provided via servers) that can be periodically downloaded into one's computers. By rigorously controlling the update process of the "stationary data", users can be confident that they have a consistent set of reference information preloaded into their computers. Adopting this approach would go a long way to ameliorating deficiency (2) "lack of unambiguous sources for standard data." This was identified above as one of the two major causes for the inability of current systems to obtain timely delivery of the three principal force package data objects (the organization structure, personnel, and materiel) with confidence in their accuracy and reliability.

2.1.2 An Organization-Centric View of Stationary Data

The other major deficiency identified above, namely, (1) "lack of a single, universal approach for information identification management," could be addressed in the armed forces by coalescing "stationary data" around the concept of force structure. Specifically, the idea is to identify the way in which military organizations (military units) are grouped, and to use these templates for linking all the other necessary resources to that basic structure. In this paper such templates for how military units relate to each other will be referred to as default operational organizations (DOO). The DOO concept is based upon three fundamental tenets.

- First, the concept of force structure lies at the heart of the representation of battle command.
- Second, a default force structure exists that is composed of a set of default organizations that are linked together with a default command structure (DCS). This force structure is relatively stable (i.e., it is "stationary data") and if designed properly (i.e., is richly populated), then it can be used as the base structure for integrating entities and building arbitrary orders of battle.
- Third, operational command structures (i.e., unit task organizations) are fluid, and are nearly always constructed by modifying (i.e., re-linking the nodes of) the default force structure.

Force structure can be formally represented using tree graphs, which are often used for organization charts. The terms organization, command structure, and unit are mapped to the formal definitions for node, a set of links, and tree, respectively. In other words, a composite unit (U) is made up of a set of organizations (O) and a set of links called a command structure (CS). The purpose of the set of links in the DCS is to define aggregation (or composition). One can interpret the set of links using the phrase "is composed of," e.g., "Battalion A 'is composed of' Company B, Company C, and Company D." The concept of DOOs states that default organization charts for composite military units do indeed exist, but to be truly useful, they must reflect the way a composite unit deploys or fights. In other words, the structure must include all the nodes typically used when the composite unit is deployed.

Currently, force structure documents are administrative in nature. In the Army, these are called Tables of Organization and Equipment (TOE), or Modified TOEs (MTOE). It is proposed that the information in these documents be enhanced by imposing a formal, hierarchical structure to it (i.e., making it into a tree graph), beginning from some arbitrary root node, say the Department of the Army, and extending down to the individual warrior level, commonly called a billet. However, to be truly useful, the structure must include the nodes that are used on a daily basis for operations, such as teams, sections, elements, and squads. Currently, these organizations are normally defined in Field Manuals that describe tactics, techniques and procedures. Therefore, an objective is to evolve the force structure documents from an administrative purpose to one that is operational in nature. Once this is accomplished, this information would constitute the "stationary data" for the ORGANIZATION domain. The "stationary data" from the other domains, e.g., MATERIEL, PERSON, etc., could be 'linked' to data in the ORGANIZATION domain.

This paper discusses a way for uniquely identifying the nodes in a DOO via globally unique enterprise identifiers (EIDs), and for using that identification scheme to label all the other types of "stationary data" that are needed to provide timely delivery of the three principal force package data objects (the ORGANIZATION structure, PERSONNEL, and MATERIEL). This would go a long way to solve the first major deficiency identified above, (1) "lack of a single, universal approach for information identification management."

2.1.3 Applicability of Proposed Approach to Logistics Stationary Data

In the context of Logistics the deficiencies mentioned above can be addressed by resolving the following two questions:

- Can the Logistics community exploit a common, detailed organizational structure, such that more efficient information identification management can result (e.g., by creating globally unique enterprise identifiers (EIDs))?

- Can the approach be realistically implemented with legacy systems so as to leverage their large sets of reference and stationary data?

This study answers the above two questions and identifies two specific ways of applying the proposed approach which could be demonstrated in the context of:

- the *building of units on the fly (BUOF)*, and

- the *integration of Logistics and Command & Control information systems.*

A summary of the proposed approach is as follows:

- base the EID scheme on the Army Organization Server (AOS) concept included in the operational requirements document (ORD) for the FMS being implemented by USAFMSA for managing military units, and

- establish an authoritative source of stationary Logistics reference and stationary data using a common *EID* scheme.

2.2 *Building Units on the Fly*

The phrase "building units on the fly" covers a variety of topics. In a Deputy Chief of Staff for Logistics (DCSLOG) study contract entitled, *Leveraging Developmental Automation Technology and Logistics Business Practices to Enhance Army's Ability to "Build Units on the Fly"*[9] four military requirements were described:

- To be able to rapidly build, deploy, and sustain full-spectrum contingency units and Task Forces (TFs),
- Maintain total visibility and fidelity of personnel and equipment,
- Ensure unimpeded and focused readiness reporting,
- Apply "drag and drop" to individual or "each" level when building a Task Force (TF) and/or Time Phased Force and Deployment Data (TPFDD).

As expected, the DCSLOG is not the only military group that requires this capability.

2.2.1 Default Operational Organizations and ORG-IDs

Given that a default force structure exists, it is now simple to enumerate the DOOs (or nodes of the tree). When a new organization is created by the force development community, it is assigned an identifier that stays with the organization for its life (i.e., until it is disestablished). This identifier is called an organization identifier, or simply ORG-ID. This allows the default force structure (tree) to be represented via a set of organizations, each with an ORG-ID (stored in an ORGANIZATION table), and a set of default links (stored in an ORGANIZATION-ASSOCIATION table), that connects every organization with a default parent, except, of course, for the root node that has no parent.

This structure can be maintained in a database, called the organization server (or org server) which is populated and maintained by the force development community. This server becomes the authoritative source of current force structure data for the organizations that it includes. ORG-IDs are EIDs, and therefore, are unique across the whole enterprise. Anyone who downloads all or part of the force structure tree will receive the same set of DOOs with their DCS.

This constitutes a formal approach for representing military organizations in a form conducive for use in military operations. Two main assertions can be made about this approach:

- First, the concept of ORGANIZATION is the central concept around which all battle command representations revolve; in other words, it forms the framework to which all

---

9 Contract Number DASW01-98-D-0060. Briefing by Col. (Ret) D. Measels, SAIC, 13 Oct 99.

10 S. Chamberlain, "Default Operational Representations of Military Organizations," Army Research Laboratory Technical Report: ARL-TR-2172; February 2000; http://www.arl.mil/~wildman/PAPERS/tr2172.html

other battle command entities and functions relate, and this can be exploited to solve several problems.

- Second, a default organizational structure exists that is relatively stable and if it is “richly populated”, it can be used as the base structure to build any desired order of battle by simply re-linking existing ORGANIZATIONS[11].

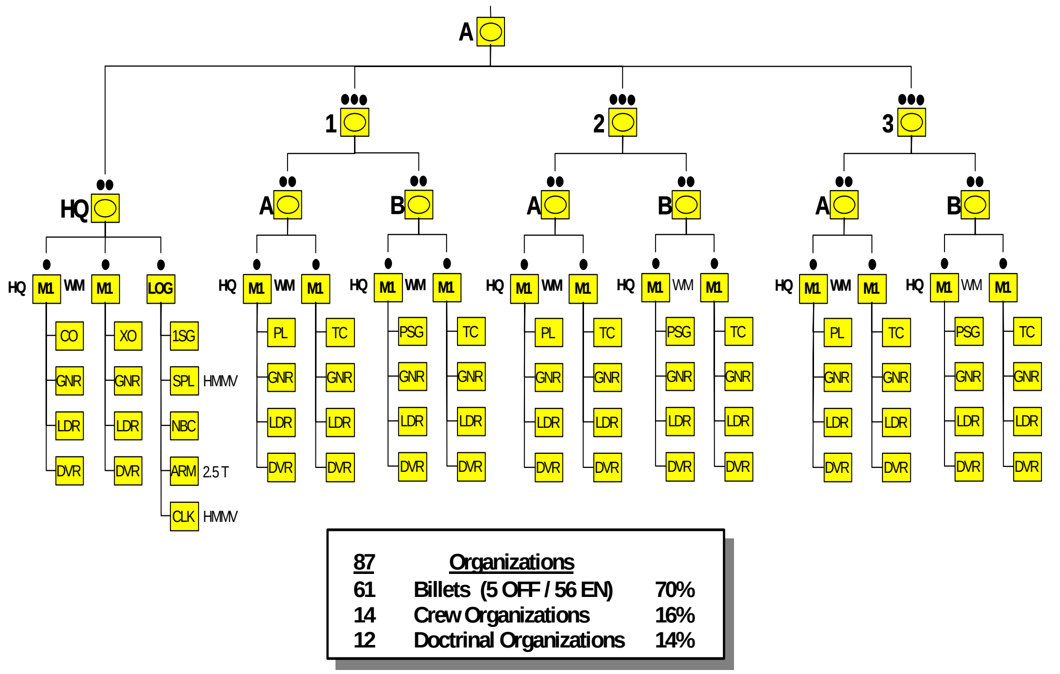


Figure 2.2.1-1: A Default Operational Organization of “A Tank Co.”

The term “richly populated” means that the default structure must reflect the way military units fight and deploy, and this will require a highly detailed structure that extends down to the individual billet level (See Figure 2.2.1-1). Therefore, individual soldiers, teams, elements, sections, squads and many other ORGANIZATIONS that have been traditionally described only in FMs and training documents must be part of the default organizational structure. In essence, what is being proposed is that force structure documents explicitly contain a higher level of detail that includes information that was previously found separately in other sources.

There are large amounts of information that can be considered "stationary data." This is information that is "semi-static," meaning that it is relatively invariant over its lifetime and can be maintained in a shared reference library. An example of semi-static data is phone numbers. Although thousands of phone numbers are added, deleted, and changed daily worldwide, it is not a frequent event for a person's phone number to change. Usually, a person's phone number is static for the duration of their tour of duty.

---

[11] This means that an organization, including a soldier, may be a member of many command structures, however, only one default version exists.

DOO information is stationary data that falls under the category of organizational information. To provide easy access to default organizational data, it would reside in an "Organization server" that:

(1) is controlled, operated, and maintained by those with inherent responsibility for force structure formulation and establishment.
(2) is maintained in a form readily usable by the warfighter (this means that the data must be directly loadable into the warfighter information systems).
(3) uses a common naming convention that cuts across service and coalition boundaries.
(4) is based upon a minimal set of design rules that cut across service and coalition boundaries.

In the U.S. Army, there is an agency whose purpose is to maintain information about the Army's force structure. This organization is the USAFMSA that is part of the ODCSOPS. In other words, USAFMSA is the authoritative source for information concerning the U.S. Army force structure. For the past year, a project has been underway by the Requirements Documentation Directorate (RDD) of USAFMSA and the TRADOC[12] Program Integration Office for Army Battle Command System (TPIO-ABCS) to build a prototype "Organization server" that contains DOOs of Army forces that extends down to the billet level. The concept is that this will ultimately serve as the definitive source of highly detailed force structure information of all Army units. Two key advantages are (1) the data is kept current by the official force data maintainers so that other agencies do not need to create their own copies, and (2) the data in the Organization server is provided using a common DoD data schema for the organization domain[13]. Therefore, both the data model and the data are openly available to any authorized user that requires force structure information.

To unambiguously identify each DOO, a universally unique identifier called an ORGANIZATION IDENTIFIER, or *ORG-ID*, is assigned to it by the force development community when the ORGANIZATION is created (See Figure 2.2.1-2). In relational database terms, an ORG-ID is a primary key; that is, it uniquely identifies the ORGANIZATION from all others. It is also a special form of primary key called a *surrogate key*.[14] A surrogate key is a primary key that has no inherent meaning; in other words, it is, for example, a number from which no information can be inferred about the data it identifies. Integers make good surrogate keys because they are simple, easy to manipulate by machines, and easy to verify for uniqueness.[15] Users never need to know that a surrogate key exists, but the database management system (DBMS) and application programs can use them extensively to greatly enhance performance and flexibility.

---

12 U.S. Army Training and Doctrine Command (TRADOC), a major four-star Army command.

13 In this case, it is the data model of the Joint Common Database (JCDB) that is used in the Army Battle Command System (ABCS).

14 Technically, a surrogate key is a primary key that has no inherent meaning, is composed of only a single (database) field, and whose value is not derived from any other entity. See Lonigro, Mike, *The Case for the Surrogate Key*; http://www.dbpd.com/vault/9805xtra.htm.

15 Technically, an identifier is just a fixed width bit pattern and not an integer. However, since it can be displayed as a whole number, the term integer is used for conceptual simplicity.

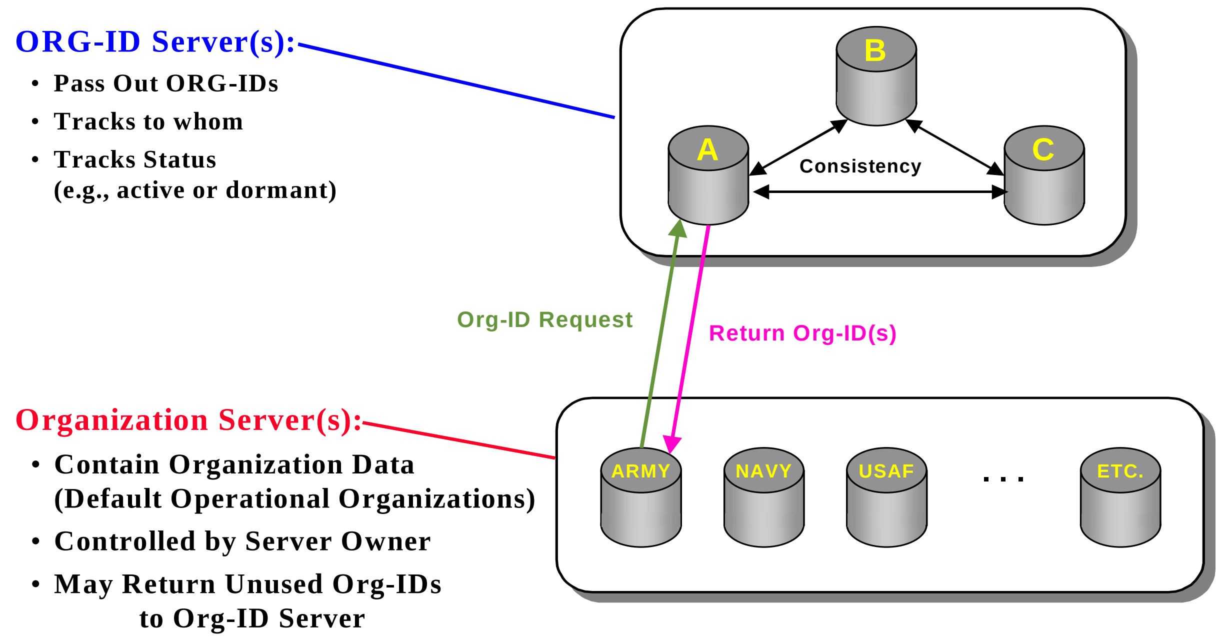


Figure 2.2.1-2. ORG-ID Servers Provide ORG-IDs For Organizations That Maintain Force Structure Data

ORG-IDs need to be unique and the implementation of a system to assign unique surrogate keys to ORGANIZATIONs should be designed to be flexible enough to operate across military service, governmental, and international boundaries. To accomplish this, a special set of independent servers, called *ORG-ID servers*, has been developed that do nothing but hand out unique identifiers. Because surrogate keys have no inherent meaning, there is no need to assign them in any order. Instead, the ORG-ID servers parcel out ORG-IDs to organization servers on a first-come, first-served basis.

All organizational information is maintained in the organization servers, not the ORG-ID servers. The only information the ORG-ID servers maintain is to which Organization server an ORG-ID was given and whether the AOS has activated the ORG-ID (i.e., whether the identifier is in use). This separation of ORG-ID servers from organization servers is important because it allows selective sharing of force structure while still participating in the process. Thus, coalition partners can obtain ORG-IDs while still keeping the data about their force structure confidential. Thus, the agencies that maintain the organization servers have complete control over the sharing of their force structure data. However, it is hoped that DoD members would normally share their force structure information with each other. Operational users of organizational data interact only with organization servers and never with the ORG-ID servers.

2.2.2 Exploitation of Default Operational Organization Data

The task of building units on the fly can be significantly simplified by the availability of highly detailed organization data. DOOs are stationary data and would be available via electronic download. Initially, distribution could be via compact disk. Normally, units would pre-load required DOOs before they deploy. This could include unit data from other services and coalition partners, although dissemination would be controlled by the unit owning the DOO data. DOOs could also be downloaded as needed if the required bandwidth is available. In any case, DOOs are another portion of a unit's stationary data.

As previously explained, a DOO can be used as the base structure for: (1) integrating disparate battlefield entities, and (2) building any arbitrary Order of Battle (OoB). The key is to base the structure of the DOO on fighting and deployment doctrine as well as the common Logistics features. By doing this, common aggregation points that are used to track military operations will be included. For example, it is a common practice in all the services to deploy weapon systems in groups (i.e., rarely alone)[16]. In the Army, a pair of tanks is called a section. Therefore, the DOO must include a section for every pair of these vehicles. If the tank system support teams of maintenance companies have a default (doctrinal) configuration, then the worse case scenario should be provided for in the DOO.

If this kind of discipline is applied throughout the force structure, then it should be a rare event to have to create an ad-hoc organization when building an OoB. Normally, existing ORGANIZATIONS are simply re-linked to build task-organized units via the standard operations order (OPORD) process. Often, they are supplied a temporary, user defined alias for the life of the Task Organization. The actual structure of a unit is determined by applying the task organization portion of an OPORD to a DOO. Therefore, arbitrary OoBs can be created and disseminated by exchanging ORG-ID pairs that identify an ORGANIZATION and a temporary parent along with any other attributes required to describe the new structure (e.g., the time interval of the Task Organization).

Every organization, down to the billet level, receives an ORG-ID that uniquely identifies it worldwide. However, this does not prevent other common identifiers from being used. Some organizations will have Unit Identification Codes (UIC), some will have DODAAC (Department of Defense Activity Address Code), and some will have one of the many other options available today. But each will have only one ORG-ID that is assigned by the force development community and receives a default parent organization.

Clearly, there will always be unforeseen situations in which temporary organizations are required to serve as ad-hoc aggregation points. Ad-hoc organizations are temporary organizations and are outside the purview of DOOs. They do not receive an ORG-ID and are not maintained in an organization server. Instead, they receive an identifier that is assigned by the unit that creates them – exactly how this is accomplished will be described later. As with other dynamic data, it is the creating organization's responsibility to disseminate any information about the ad-hoc organizations it creates to all required recipients before they are used.

BUOF is a primary reason for developing DOOs. By providing a highly detailed, richly populated, default organization structure from which to begin the process, the use of the "drag and drop" paradigm[17] to build new units from existing ones is easily facilitated. If other entities

[16] The services even share the common terms "Lead" and "Wingman."
[17] Analogous to file movement in Microsoft Windows98™ operating systems.

from the data model of the Joint Common Database (JCDB)[18] standard are used, then a myriad of other information can also be managed. Good examples that are pertinent to Logistics applications are its variety of location representations and the relationships defined with three other primary battlefield domains: ORGANIZATION, MATERIEL and FACILITY.

An ORGANIZATION is defined as: an administrative structure with a mission; it encompasses not only civilian but also military organizations, particularly, MILITARY-UNITs such as for example XVIII Airborne Corps, 82nd Airborne Division, 101st Airborne Division, 3rd Infantry Division (Mech.), 10th Mountain Division, 2nd Armored Cavalry Regiment. A FACILITY is defined as: real property, having a specified use, that is built or maintained by people; a typical example is a particular building (e.g., Bldg. 1234). MATERIEL is defined as: an object of interest that is non-human, mobile, and physical; a typical example is a particular battle tank (e.g., tank C-8). ORGANIZATIONs, FACILITIES, and MATERIEL may have geographic locations (e.g., latitude and longitude), while ORGANIZATIONs and FACILITIES may have addresses, both physical and electronic. An ORGANIZATION may "occupy" a FACILITY and "employ" MATERIEL. This structure provides several options for specifying the LOCATION of an ORGANIZATION using different identifiers. For example, ORGANIZATIONs can be located using geographic coordinates for themselves or from the FACILITY or MATERIEL that they are using. An ORGANIZATION can have its own physical address (e.g., street, city, and state), or it can inherit one from the FACILITY that it is occupying. ORGANIZATIONs can have alternative logical identifiers, like UIC and office symbols, alternate addresses, like a DODAAC, and electronic addresses, like host or domain names, IP addresses, telephone numbers, and e-mail addresses. Therefore, combining DOOs with other basic stationary data establishes significant flexibility and alternatives for the location of organizations and assets in the Logistics domain.

The concept of ORGANIZATION serves as the integration point for all other battlefield entities. Therefore, as one "drag and drops" DOOs in the process of building units on the fly, all the data associated with the DOOs can also be easily identified and collected. This facilitates the Logistics planning process as options and alternatives that can be easily configured for evaluation. However, the DOOs are just the starting point. The associated data, such as FACILITY and MATERIEL data, must also be easily identified and collected. This is the topic of the next section.

2.3 Integrating Logistics and C2 Information

2.3.1 The Basic Technology: Enterprise Identifiers

In any information system, a critical feature is the ability to link together disparate pieces of data and information via relationships. One way to greatly facilitate this task is to provide a common technique for identifying the pieces so that they can be conveniently referenced. This is the objective of EIDs. If all data is referenced using a common format, then one can effortlessly "plug and play" data and information. In the Internet community, this is being accomplished

[18] The schema for the Joint Common Database (JCDB) that is employed in the Army Battle Command System (ABCS) being developed by the PEO C3 Systems, Ft. Monmouth, NJ.

with the Uniform Resource Identifiers (URIs).[19] However, for databases, and particularly databases communicating using low bandwidth environments, the approaches being used for URI are verbose and of highly variable sizes. Consequently, a type of URI for database applications is required.

The term *enterprise key* (EK) comes from the relational database community[20]. An EK is a type of EID. It is a *surrogate key* (i.e., a primary key with no intelligence built into it), which is an important characteristic for integrating disparate data systems. What makes an EID special is that it uniquely identifies an entity (or object) *across the entire* enterprise, not just within a specific data table.[21] For example, if the enterprise were the DoD, then under this scheme no two data items would have the same EID within the DoD. Thus, if a piece of data is entered with an EID, no matter where that data propagates, it is guaranteed to be uniquely tagged. Similarly, if two data items have the same EID, then they must be semantic equivalents, that is, they must represent the same thing. This allows users to use different information management approaches to manage identical items. It would not matter if the item is represented in a relational database in one system (with an EK) and in another as an object (with an Object ID); if they are the same EID, then they are semantic equivalents.

EIDs are designed for computers, not users. Users need not be aware that EIDs exist; in fact, they may continue to use the keys with which they are familiar, called "business keys." For example, a person may or may not have a social security number (SSN), a value that is often used as a primary key. Similarly, an instance of MATERIEL-TYPE may or may not have a national stock number (NSN), another value that is often used as a primary key. However, EIDs can be used to identify all instances of PERSON and MATERIEL-TYPE regardless of whether they have SSNs or NSNs; the domain they represent does not matter because EIDs are just numbers without special meaning assigned to identify instances of each given entity. If SSNs and NSNs are available, which may be the norm, then they can still be used to uniquely identify records within those entities. Thus, systems can maintain their own identifier system in conjunction with EIDs and EIDs can be relegated to intersystem operations.

---

[19] For URI Working Group Charter, see http://www.ietf.cnri.reston.va.us/proceedings/94mar/charters/uri-charter.html.

[20] The relational database term "key" is a specific implementation of the generic concept of "identifier."

[21] Tom Johnston has written an excellent set of articles on EKs for DM Review that are available online. The first two articles describe the basic problem and the last four discuss implementation issues:
Primary Key Reengineering Projects: The Problem; DM Review, February, 2000;
http://www.dmreview.com/master.cfm?NavID=55&EdID=1866
Primary Key Reengineering Projects: The Solution; DM Review, March, 2000;
http://www.dmreview.com/master.cfm?NavID=55&EdID=2004
De-Embedding Foreign Keys, Part 1: DM Direct, June 2, 2000.
http://www.dmreview.com/editorial/dmreview/print_action.cfm?EdID=2308
De-Embedding Foreign Keys, Part 2; DM Direct, June 9, 2000.
http://www.dmreview.com/editorial/dmreview/print_action.cfm?EdID=2322
De-Embedding Foreign Keys, Part 3: DM Direct, June 16, 2000.
http://www.dmreview.com/editorial/dmreview/print_action.cfm?EdID=2331
De-Embedding Foreign Keys, Part 4: DM Direct, June 23, 2000.
http://dmreview.com/editorial/dmreview/print_action.cfm?EdID=2341

A major advantage can be realized if all data entities (across the enterprise) use EIDs of a common format, i.e., there is one format for primary keys that is used for all entities within the enterprise. This allows one to arbitrarily link together any two pieces of data without having to know the structure of the data ahead of time. This will be a primary enabler in the process of integrating disparate systems. Once an item is uniquely identified, other tools and standards can be used to obtain information about the structure of the data so that it can be assimilated.

For example, if the data schema is described using XML,[22] then the data structure information is unambiguous and easy to obtain.

The use of enterprise identifiers presents an interesting issue in the relational database environment. The reader is directed to the detailed analysis and comments presented in Appendices D and G for further clarification. Briefly stated, the implications of using EIDs are as follows. Recall that EKs are surrogate keys and, therefore, are composed of only a single (database) field whose value is not derived from any other entity. This means that if an EK is a primary key, then the primary key is a single attribute that is not derived from any other entity. In other words, the table is independent. If all primary keys are EKs, then all the tables in the database can be implemented as independent tables. Database designers and engineers may object to this policy because it limits some shortcuts in the query process and can be construed as requiring unnecessary space. However, this is a small price to pay when one considers the enormous benefit of obtaining the complete and unencumbered flexibility and expandability provided by EIDs. The various aspects of this debate among the relational database theorists and practitioners is covered thoroughly in the articles listed in footnote 21. Further, many of the objections presented about EKs disregard the fact that most queries and database operations are not executed by humans, but by sophisticated applications programs that can exploit the features of EKs to significantly improve database performance and reduce overhead.

2.3.2 Enterprise Identifiers Servers and DOOs

The primary challenge in creating an EID system is developing an approach to guarantee that EIDs are unique while ensuring that they can be easily obtained. When a database management system needs to create a piece of data, it must call an *Enterprise Identifier server*, or *EID server*, to obtain an EID. The goal is to develop an extremely flexible scheme for assigning EIDs that allows as much control or freedom as the system managers require.

One approach is to link the EID assignment system with the DOO concept, that is, base the EIDs on ORG-IDs. Because ORG-IDs are universally unique (i.e., are another kind of EID), concatenating another unique value to it also results in a universally unique identifier[23]. If an EID Server is aligned with an ORGANIZATION, then all it must do is maintain a list of unique

[22] XML – The Extensible Mark-up Language, a standard for describing the structure of information. Proposed standard ways for capturing the structure of the data are being developed based on XML, such as XMI (XML Metadata Interchange). In addition, large software producers are adopting XML as the data transfer mechanism (see for instance the recent announcement by Microsoft to implement XML as the intermediary for exchanging data among all applications in the next release of its developer suite dubbed Visual Studio.NET). See http://www.w3.org/XML/.

[23] This is a common approach used for a variety of applications, for example, to define XML name spaces.

values that it concatenates to the ORG-ID of the ORGANIZATION it represents. If an ORG-ID is 32 bits in length, then 4.3 billion ORGANIZATIONs can operate that number of EID Servers. If the second, concatenated value is also 32 bits in length, then each EID Server can allocate 4.3 billion EIDs. In other words, 4.3 billion EID servers can hand out 4.3 billion EIDs for a total of 18 billion-billion EIDs. All EIDs would have the same format, a 64-bit identifier (See Figure 2.2.3.2-1).

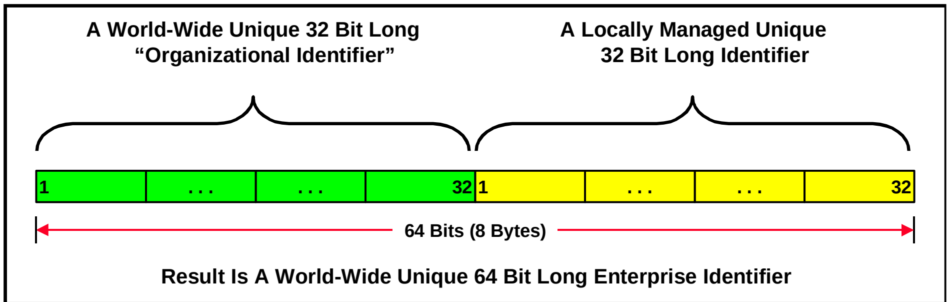


Figure 2.2.3.2-1. A World-wide, Unique Enterprise Identifier Can Be Composed By Combining Two Unique Numbers.

The advantage of this approach is that it can be as centralized or distributed as required by the implementers. This is because ORG-IDs are a form of EID that the force development community passes out as an administrative procedure ahead of time. Therefore, every official organization has a pre-allocated EID in the form of its ORG-ID. Imagine two extreme cases. At one extreme, one centralized computer could represent 4.3 billion virtual EID Servers, one for each possible ORGANIZATION[24]. At the other extreme, there can be 4.3 billion distributed machines each with a resident EID Server. In practice, the real configuration will lie somewhere in the middle. This flexible approach allows ORGANIZATIONs to operate isolated C2 systems, like those found on the front lines of the battlefield, with their own on-board EID Servers, while ORGANIZATIONs in friendlier territories, with high bandwidth communication links, may link their C2 systems to central servers. The choice is completely up to the system managers. All that an ORGANIZATION needs to establish an EID Server is to be registered with its service's force development community via an entry in the service's Organization server. This results in being assigned an ORG-ID. Once that is completed, it can create EIDs that are guaranteed unique within the enterprise, in this case, the DoD (see Figure 2.2.3.2-2).

24 Although this is an impractical option due to the bottleneck it would produce, it is technically feasible. The server would only have to maintain a list of the last value allocated for each ORG-ID. There are 4.3 billion 64 bit (8 byte) EKs which amounts to 34.4 giga-bytes, a file size that can be easily handled by a personal computer.

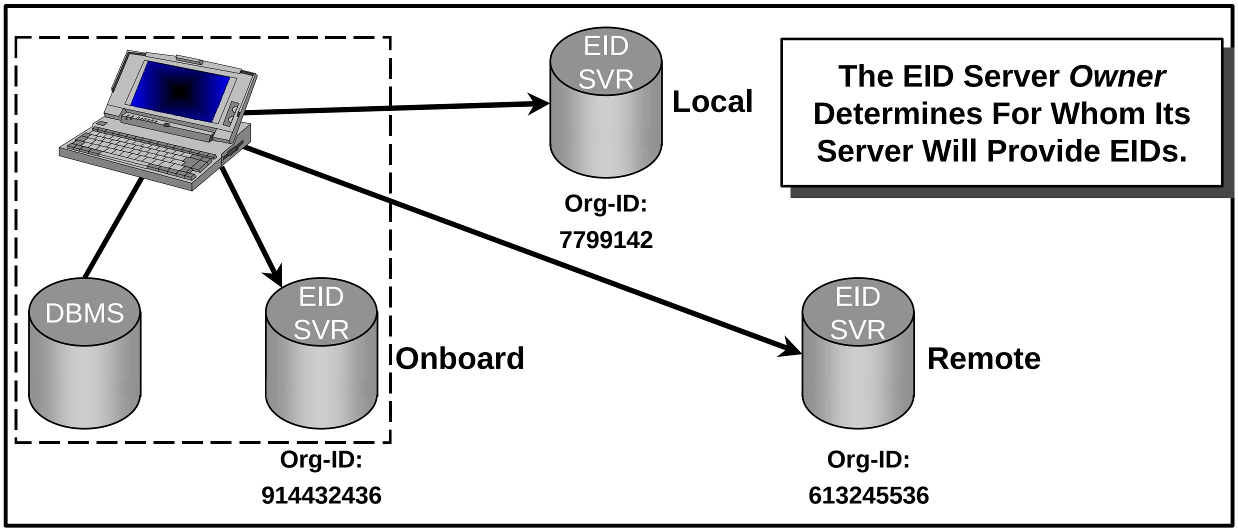


Figure 2.2.3.2-2. An EID May Be Requested From Any EID Server That Allows It

2.3.3 The Materiel Server

Just as an Organization server contains stationary data about the Army's default force structure, a Materiel server can be developed to serve as the definitive source for MATERIEL related stationary data. As with the Organization domain, the Materiel domain in the JCDB data model consists of two basic categories of entities: MATERIEL and MATERIEL-TYPE. MATERIEL represents actual physical objects that usually have a serial number or other identification. These are typically part of an ORGANIZATION's property book. MATERIEL-TYPE represents classes of objects that usually have NSNs to identify them. In database terms, a piece of equipment such as a tank is an instance of MATERIEL, which is categorized as belonging to a given class of materiel via MATERIEL-TYPE (see Figure 2.3.3-1).

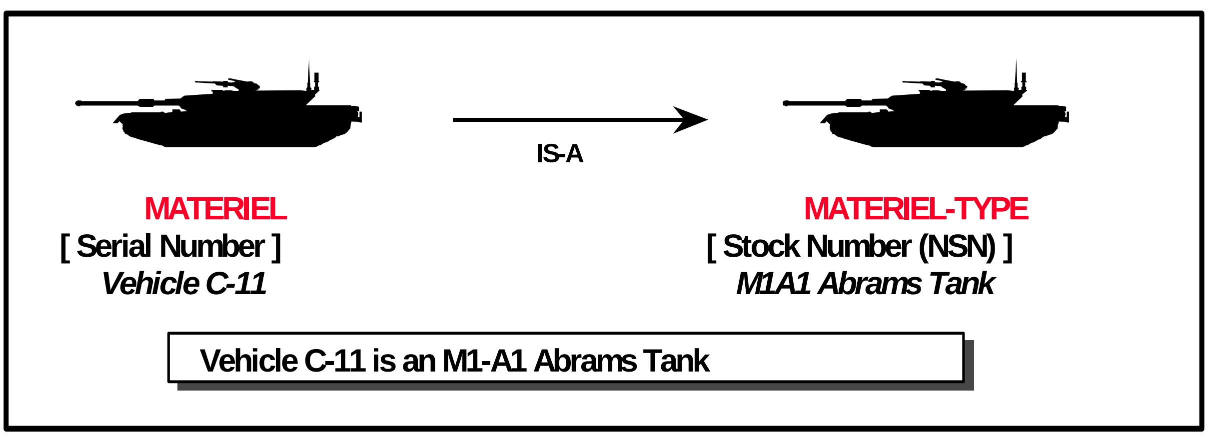


Figure 2.3.3-1. Distinction Between MATERIEL and MATERIEL-TYPE

Initially, a Materiel server would contain only MATERIEL-TYPE information that is created and maintained as cataloging data by the Defense Logistics Agency's (DLA) Defense Logistics Information Service[25]. Clearly, this information already exists digitally in various forms throughout the DoD. But the challenge for a Materiel server is to maintain the Logistics data in a form that is readily usable directly by the warfighting systems. This requires that MATERIEL-TYPE data in the server be maintained in a standard DoD schema for MATERIEL-TYPEs (i.e., the JCDB data model). Consequently, the features of a Materiel server are analogous to those of an Organization server. Like the Organization server, a Materiel server:

(1) is controlled, operated, and maintained by those with inherent responsibility for MATERIEL-TYPE establishment, formulation, and tracking.

(2) is maintained in a form readily usable by the warfighter (this means that the data must be directly loadable into the warfighter information systems).

(3) uses a common naming convention that cuts across service and coalition boundaries (i.e., an EID system), and

(4) is based upon a minimal set of design rules that cut across service and coalition boundaries.

DOO based EIDs can significantly facilitate the task of simultaneous distributed population of a Materiel server. Recall that *any* ORGANIZATION may establish an EID server in its name and that an EID server may reside on any machine. The ORGANIZATION may be a billet or a top-level ORGANIZATION (e.g., ODCSLOG). Further, when a database management system (DBMS) executes an insert operation to create a new record, it must assign a primary key to the record. Using an EID system, the DBMS must contact an EID server to obtain a new, universally unique EID for the primary key. The EID Server it contacts may be resident on the same machine as the DBMS, or it may be located somewhere on the network. Further, the EID server may be hosted at any ORGANIZATION the data manager has authorized. It is perfectly permissible for each table in the DBMS to be configured to contact a different EID server when a new record is created for that table. Just how the system is configured is a management decision based upon performance and data ownership and is completely flexible.

The only goal is to ensure that no two EIDs collide, and each EID server does this independently[26].

---

25 See the Defense Logistics Information Service cataloging site: http://www.dlis.dla.mil/dla/index.htm

26 Note that the scenario depicted above is just one of many. An equally valid approach at the time of creating record for stationary and reference data sets without having to contact the pertinent EID Server(s) for each Create/Insert operation is to hand out a block of EIDs to the database manager that can be looked up and used locally. Data base managers can readily estimate the number of records for stationary and reference data they intend to create, and, therefore, can state with reasonable accuracy how big is the block of EIDs they require to carry out their functions. Note also that the Army could have a large number of these EID Servers, and each could hand out globally unique EIDs without difficulty or risk of ever running out of EIDs to distribute. But even if one imagines a situation where all thousand or more EID servers have exhausted their respective sets of 4.3 billion EIDs, all the Army would need to do is just to authorize the use of yet another thousand ORG-IDs as the seed for the next thousand sets of 4.3 billion EIDs. Though it is not impossible to imagine that these large sets of EIDs may be used up due to extremely inefficient key management schemes, it is arguably unlikely that all 18 billion billion EIDs will be assigned in the first implementation of the scheme. Were this to happen,

The simplest, best performing configuration is when the EID server is resident on the same machine as the DBMS, and it is representing the billet (an ORGANIZATION) of the person creating the data (e.g., recall the on-board case in Figure 2.2.3.2-2 above). This would be the typical case for a front line warrior operating a fire control system. However, for administrative tasks like populating a Materiel server in a benign, well connected environment, the options are endless. Four extreme examples are illustrated in Figures 2.3.3-3 through 2.3.3-6. These options are described in Figure 2.3.3-2.

	Materiel Server	EID Server	
Figure 6:	*Centralized*	*Centralized*	**One, Collocated Server**
Figure 7:	*De-centralized*	*De-Centralized*	**Many, Collocated Servers**
Figure 8:	*Centralized*	*De-Centralized*	**One and Many**
Figure 9:	*De-centralized*	*Centralized*	**Many and One**

Figure 2.3.3-2. Four Extreme Cases for Materiel and EID Server Configurations

For example, say Organization X has responsibility for the materiel server. Organization X is large and its DOO contains several hundred ORGANIZATIONs composed of directorates, divisions, branches, teams, and ultimately billets (i.e., people). In the simplest case, one person enters all the data to populate the Materiel server – a highly unlikely situation. Instead, many different people may populate the server based upon the type of the data. One division (X1) may handle aircraft with one branch for rotary wing systems (X1R) and one for fixed wing systems (X1F). Another division (X2) handles armored vehicles with one branch for tracked vehicles (X2T) and one for wheeled vehicles (X2W). In this case, there are many options for how the EID system may be configured, all equally plausible, but some perhaps better performing than others.

Here are just a few of the intermediate configuration options:

(1) Everyone uses one DBMS and one EID server. Therefore, everyone logs onto one DBMS and every table gets its EID from a locally resident EID server that represents ORGANIZATION X. This configuration is illustrated in Figure 2.3.3-3.

(2) Everyone in the aircraft division uses one DBMS and everyone in the armored vehicle division uses another DBMS, but both DBMS' get their EIDs from the same EID server that

---

nothing prevents the scheme to adopt a larger key structure. In fact, Microsoft Access 2000 already supports a data type based on a 128-bit long integer to be used in database replication operations. This set contains 3.4 x 10^{38} unique numbers sufficient to support even the worst case scenario if one recalls that the total number of subatomic particles in the whole universe is estimated to be on the order of 10^{80}.

represents ORGANIZATION X. This configuration is an example of the general case illustrated in Figure 2.3.3-6.

(3) Same as (2), and each division (X1 and X2) has an EID server that serves its DBMS. This configuration is an example of the general case illustrated in Figure 2.3.3-4.

(4) The same DBMS configuration as (2) and (3), but every branch has an EID server for X1R, X1F, X2T and X2W. This configuration is an example of the general case illustrated in Figure 2.3.3-5.

All these options (and many others) are equally plausible and provide unique EIDs. The location of DBMS' and EID servers are independent, as is the choice for the number of EID servers and distributed DBMS. This flexibility is the advantage of a DOO based approach.

But what if the divisions are at different geographical sites, perhaps across service or coalition boundaries? Now performance and security issues may drive the design. For the best performance, a DBMS needs to have excellent connectivity with the EID server(s) to which it must connect. This may require them to be collocated on the same machine, which does not present a problem since DOOs extend down to the individual billet level. The local EID server can represent any of a number of DOOs. Security may also require compartmentalization of DBMSs. Once again, this does not cause a problem since EID servers can be compartmentalized with the DBMSs. In both of these cases, the independence of DBMSs and the EID servers allows a myriad options to be implemented.

As a side issue, it must be understood that using EIDs does not magically solve the data fusion problem. There is no way to prevent two people from entering duplicate data into two different systems. For example, nothing can prevent Army force developers from entering Navy Carrier Groups into the AOS, other than the fact that policy ought to prevent them from doing it. The same is true for a Materiel server. Policy must be developed to prescribe who is to control (i.e., who will have the authority to create and maintain) what data. There is nothing that can physically prevent two individuals from entering duplicate (or near duplicate) data about the same class of MATERIEL, using two different EIDs into a DBMS.

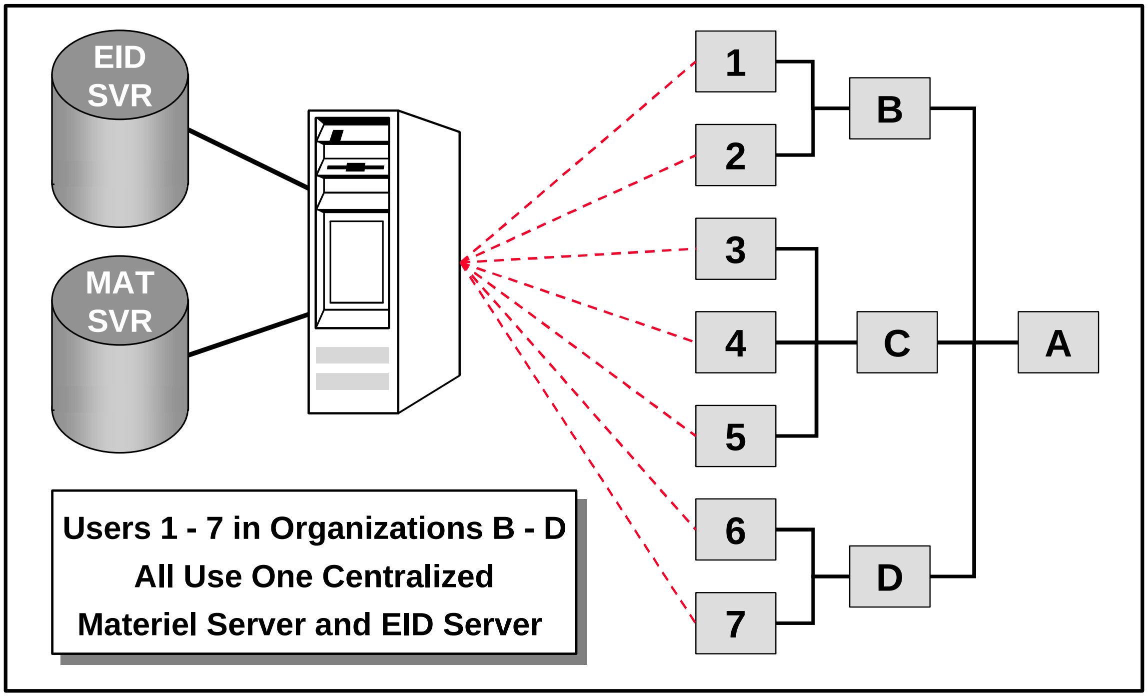


Figure 2.3.3-3. Extreme Option – Completely Centralized Control

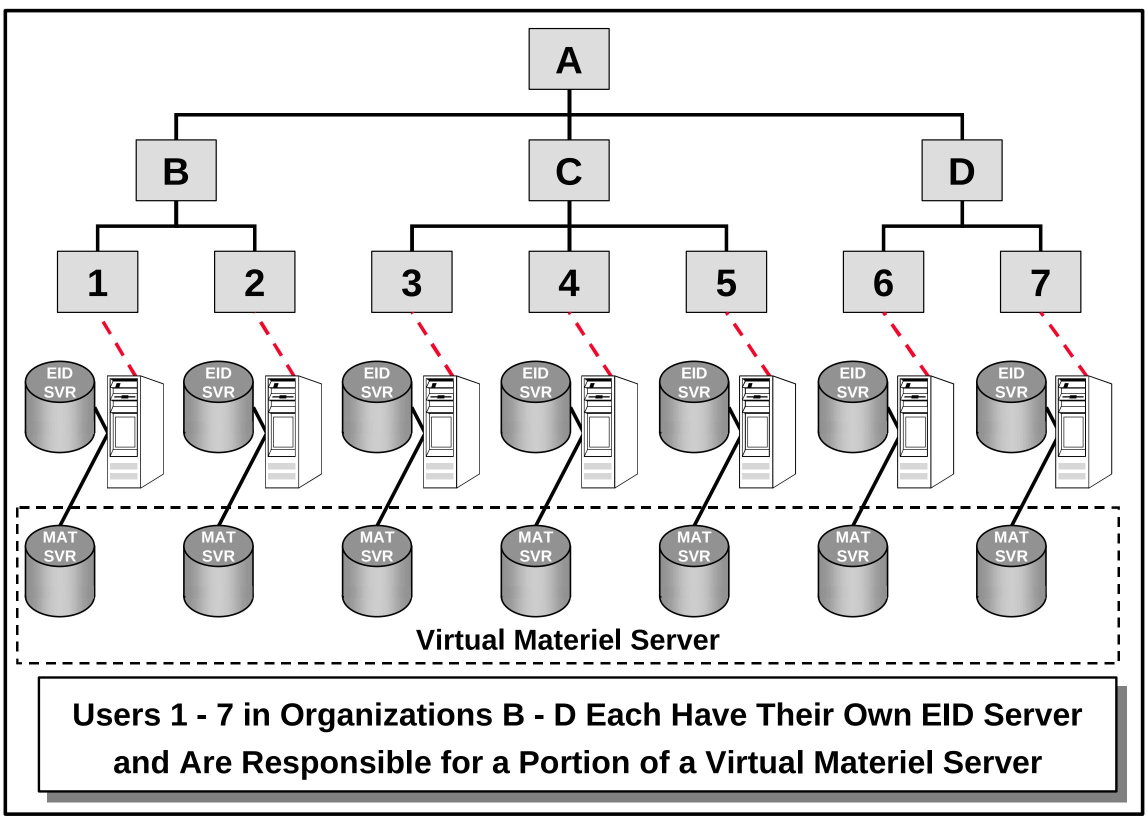


Figure 2.3.3-4. Extreme Option – Completely Decentralized Control

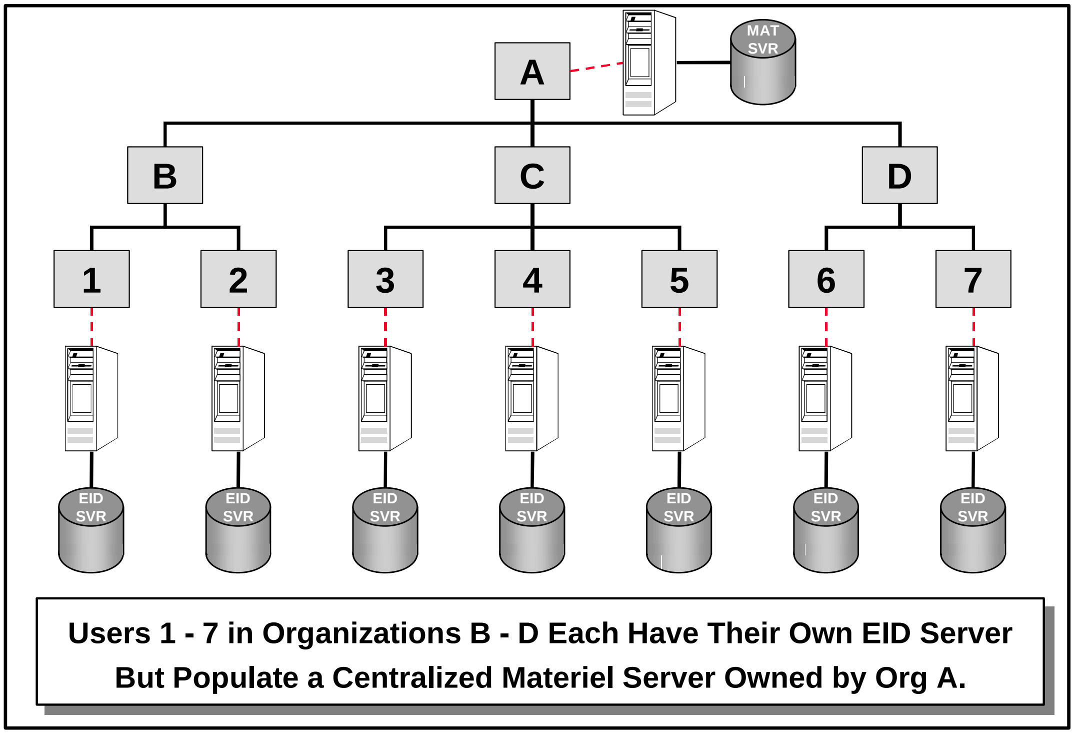


Figure 2.3.3-5. Extreme Mixture 1 – Centralized Materiel Server, Decentralized EID Server

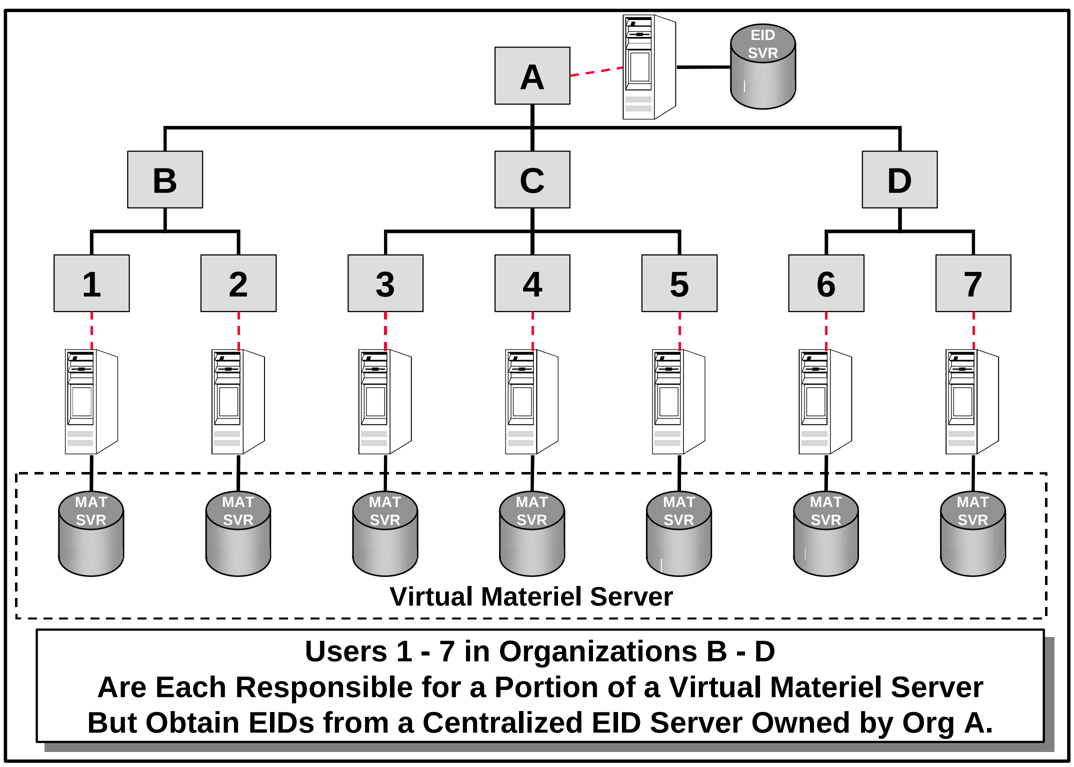


Figure 2.3.3-6. Extreme Mixture 2 – Centralized EID Server, Decentralized Materiel Server

A solution to this problem is outside the EID server capability, although having universally unique EIDs can facilitate a solution to this predicament. But in any case, policy for who is responsible for what data is imperative to a sound solution.

An even more difficult, long-term challenge is the development of a common, detailed taxonomy for the Materiel domain. It is required to provide higher levels of abstractions to represent battlefield entities in battlefield systems. The development of this second capability is a major undertaking that is insidiously arduous. Although the current system includes some levels of hierarchical decomposition, more detail is required. The problem is that there is no unique taxonomy that satisfies all purposes; a solution will be based on consensus. Therefore, a realistic solution must include distributing the task across functional areas. For example, one approach would be to allow each of the Army's Integrated Materiel Management Centers (IMMCs) to develop a taxonomy for those items under their purview. This would be more complex for items with no single proponent that extends across services, for example, helicopters like the Sikorsky H-60 series that is employed by all the services. Someone must be put in charge to coordinate the development of a taxonomy across service boundaries.[27]

2.3.4 Organization Server and Materiel Server Integration

A large benefit would emerge from the synergistic effects of combining the Organization server data with that from the Materiel server. If this were to be implemented, then a highly detailed, direct link could be established between these two major domains in a common format that could be directly accessed by any Army battlefield system. To take full advantage of a common schema, an arbitrarily scalable approach must be employed in the linking of these two domains.

Because DOOs are maintained down to the billet level, MATERIEL data (and data from other entities) could be linked at any level in the DOO, from personal equipment at the billet level, to large assets like vehicles, aircraft, and ships at aggregation points called *crews*. Arbitrary aggregation points, called *doctrinal organizations*, could be included any place where the force developers require (e.g., ORGANIZATIONs like platoons, battalions, and squadrons). The synergy occurs as the nodes are used as aggregation points for a wide variety of entities. This includes personnel, networks, plans, objectives, and targets that can all be linked together by referencing the ORG-IDs of DOOs.

The Organization and Materiel servers provide the basic elements to integrate C2 and Logistics reference and stationary data. Once a Materiel server is completed, the Organization Server can be enhanced to include information about authorized amounts of MATERIEL-TYPEs. A direct link can be made between the servers so that information can be continuously exchanged using a standard DoD data schema. A common, digital Table of Organization and Equipment (TOE)/MTOE becomes a reality.

However, the true payoff is the capability to download accurately maintained stationary data to the tactical units in the field (e.g., ABCS and GCSS-A[28]). This capability allows large amounts

---

27 See also Appendix F below on some of the alternative treatments for the Materiel domain taxonomy issue.

28 The Global Combat Support System – Army: see http://www.peostamis.belvoir.army.mil/index3.htm

of stationary data to be cached locally in any database before deployment. Just as a basic load of ammunition is allocated, a basic load of information is preloaded into a unit's databases. Every database has a stockpile of relatively static stationary data with common EIDs. This alleviates the requirement to download most stationary data during operations, thus saving communication resources for real-time operational data. Since the stationary data contains a common set of EIDs that is shared by all users, two key advantages result. First, the terse EIDs can be exchanged rather than the data itself to significantly reduce unnecessary bandwidth usage. Second, because the keys reference semantically equivalent entities, it doesn't matter exactly how the data are managed in each local database. For example, they may be stored as relational database entities or as object database objects. Further, text strings (e.g., descriptive or explanatory text) associated with a given record or instance of an object may be stored in different languages, thus reducing the probability of misunderstanding between coalition partners.

Using the cache of stationary data with common EIDs, tactical units are free to re-link the stationary data as required to build orders of battle, plan and conduct operations, and exchange Logistics requirements and status. This can be accomplished efficiently using the operations order and situational awareness processes already in place. But because each system has a common set of Logistics reference and stationary data pre-loaded into its database, the coordination required to disseminate the changes is greatly reduced. Links to data (i.e., EIDs) can be exchanged rather than the data.

3.0 IMPLEMENTATION STRATEGY

3.1 Side Issues and Distractions

3.1.1 C3I and Logistics Stovepipes

There are other issues that complicate the implementation process for both requirements, building units on the fly and the integration of Logistics and C2 systems. First, many other domains share both of these tasks. Therefore, the Logistics community must depend on others to participate in the solution. This factor always increases the difficulty of a task. Second, the C2 and Logistics communities have a history, perhaps longer than any other - including the intelligence community - of maintaining separate domains with parallel systems. The stovepipes begin at the highest DoD levels where the Under Secretary of Defense for Acquisition, Technology and Logistics (USD(AT&L))[29] maintains the Logistics function and the Assistant Secretary of Defense (Command, Control, Communications, Intelligence) (ASD(C3I))[30] has the intelligence and C2 functions. The Defense Information Systems Agency (DISA)[31] and the Defense Intelligence Agency (DIA)[32] are in one stovepipe and the Defense Logistics Agency (DLA)[33] in another. Therefore, to be successful, a solution must involve cooperation from the highest levels of the DoD. A solution derived at any lower level will simply solve a subset of the problem without a significant, long-term impact. These are inherently joint requirements.

3.1.2 Different Standardization Schemes

Another issue surrounds the information standardization process. Currently, there are two subtle competing implementation schemes to achieve standardization. The first is based on the notion that everyone should use a common data schema. This is reflected in the DoD 8320.1 Data Standardization Procedures[34]. The second approach asserts that there should only be a standard way of standardizing; that is, there should be a standard way of describing one's information and processes so that anyone can obtain formal and unambiguous descriptions that will allow them to interface with a system. This is reflected in the DoD Defense Information Infrastructure Common Operating Environment (DII-COE) procedures. A good example is the Shared Data Environment (SHADE) procedures.[35] Both approaches are valid, have advantages and

---

29 Under Secretary of Defense for Acquisition, Technology and Logistics; see: http://www.defenselink.mil/pubs/almanac/almanac/organization/USD_Acquisition.html

30 Assistant Secretary of Defense (Command, Control, Communications and Intelligence); see: http://www.dtic.mil/c3i/

31 Defense Information Systems Agency; see: http://www.disa.mil/disahomejs.html

32 Defense Intelligence Agency; see: http://www.dia.mil/

33 Defense Logistics Agency; see: http://www.dla.mil/

34 DoD 8320.1-M-1, *Data Standardization Procedures*, dated April 1998; see: http://www-datadmn.itsi.disa.mil/8320_1m1.html

35 SHADE; see: http://dii-sw.ncr.disa.mil/shade

disadvantages, and most importantly, are realistically required. However, having two approaches can lead to problems because the results produced can cause conflicts when one approach is expected, but the other is used. This study has concluded that a compromise or combination of these two implementation approaches is a pragmatic way to achieve the desired goals.

3.1.3 The Golden Rule of Data

They who have the data make the rules.

The golden rule of data is a double-edged sword. From one perspective, this is precisely the cause of the data interoperability problem – every owner of data is developing its own standard. The result is excessive duplication or different meanings for common terms that require individual translation interfaces to be built causing a combinatorial explosion. This is an intractable situation and must be prevented. From another perspective, producing data standards independently from the data owners and maintainers may be a futile endeavor. It is the data that is sought after, not the model. Therefore, when databases already exist that are populated with large quantities of data, it is imperative that the data standards related to this data be developed in congruence with it. Although compromise is inevitable, the standardization process must be closely aligned with the existing data and its maintenance procedures. Consequently, the data modeling community must strive to establish close working relationships with the owners of large existing databases and obtain guidance from them.

3.2 Servers for Stationary Data

The basis of the recommended implementation plan revolves around the concept of stationary data servers. Recall that stationary data is semi-static information that can be stored in a repository, normally called a server, to provide easy access to the data. To exploit the advantages of both data standardization schemes, it is recommended that a core subset of the DoD common data schema become the focal point of a venture to provide common data for battlefield systems via a set of domain servers. Several DoD data models (Land C2 Information Exchange Data Model (LC2IEDM), Army Integrated Core Data Model (AICDM), JCDB, Core Architecture Data Model (CADM))[36], all related to the C2 Core Data Model (C2CDM),[37] include a set of five core battlefield domains. The domains are *organization, materiel, person, feature, and facility*. In this study, the emphasis is limited to the *organization* and *materiel* domains.

---

[36] AICDM: Army Integrated Core Data Model; see:
http://aodr-arch-odisc4.army.mil/DataShopper/login.asp (password Protected)
CADM: C4ISR Architecture Data Model; see
http://www.c3i.osd.mil/org/cio/i3/AWG_Digital_Library/index.htm
JCDB Data Model; see:
https://peoc3s.monmouth.army.mil/homepeoc3s.nsf/Home?OpenFrameSet (password protected).
LC2IEDM: *Land C2 Information Exchange Data Model*, 1 Oct, was Army Tactical Command & Control Information System (ATCCIS) Generic Hub 4: ADatP-5 Draft 1.0; see
http://www.euronet.nl/users/atccis/index.html

[37] Command & Control (C2) Core Data Model; see:
http://www-datadmn.itsi.disa.mil/ddm.html.

To implement both requirements (building units on the fly and the integration of Logistics & C2 systems), sources of stationary data must be identified that can support the modeling, standardization, initial population, and most importantly, continuous maintenance of the information. The way to realistically achieve the maintainability criteria is to *identify an authoritative source of the information that already has data maintenance as part of its responsibility*. Fortunately, significant amounts of Logistics reference and stationary data are available in digital form, although much of it is not in an enterprise-wide, common schema.

What is required is a change in *modus operandi* to change the focus from the data modeler to the data maintainer. Simply put, a data model without data is like an engine without fuel[38]. *Although a data model may be required, it is not sufficient.* Therefore, the ultimate goal of the data standardization process should continue beyond the data modeling aspects of the problem to include the process of populating and maintaining standard data.

Recommendation 1: In the data standards community, a more aggressive goal should be established to produce a common data model *and* associated databases of standard stationary data.

3.3 Building Units on the Fly

USAFMSA is the authoritative source of stationary ORGANIZATION and ORGANIZATION-TYPE data. Creation and maintenance of an Organization (Org-) Server is included as part of their FMS project's ORD. The Org-Server will provide organizational information down to the billet level. This includes those doctrinal and crew organizations that are required to conduct operations. For example, if there is a standard way to organize combat repair teams into sub-units, then this should be reflected in the DOO for a forward support company.

Recommendation 2: The TSM-CSSCS[39] should work with the Logistics TOE builders in the Training and Doctrine Command (TRADOC) Logistics schools and USAFMSA (Requirements Documentation Directorate (RDD)-Lee) to ensure that the DOOs for Logistics units include the doctrinal organizations required for typical deployment options.

Recommendation 3: The DCSLOG should strive to identify duplicative efforts within the Logistics community that independently designate organizations and populate organizational databases and work closely with the DCSOPS to define a path forward to ultimately converge the efforts of USAFMSA with those duplicative efforts within the Logistics community.

---

38 Quote from LTC Robert Hartel, U.S. Army TRADOC Program Integration Office for the Army Battle Command System (TPIO-ABCS).

39 TRADOC System Manager for the Combat Service Support Control System.

3.4 Logistics and C2 System Integration – Materiel Server

Just as an Organization server stores reference and stationary data about the Army's default force structure, a Materiel server should be developed to serve as the definitive source for stationary materiel data. Clearly, this information already exists digitally in various forms throughout the DoD. However, to make the data directly accessible to automated Army battle command systems, the data must be converted to a standard schema. The schema used by the ABCS, to include the Combat Service Support Control System (CSSCS), is the JCDB data model.

Recall that data from the Materiel domain is composed of two basic entities: MATERIEL and MATERIEL-TYPE[40]. MATERIEL represents actual physical objects that usually have a serial number or other identification. These are typically part of an ORGANIZATION's property book. MATERIEL-TYPE represents classes of MATERIEL that usually have national stock numbers to identify them. Initially, a Materiel server would contain only MATERIEL-TYPE information that is created and maintained as cataloging data by the DLA Defense Logistics Information Service (DLIS). Ultimately, both MATERIEL-TYPE and actual MATERIEL data should be included, and would be entered at the point the instance of MATERIEL officially enters the Army inventory (i.e., is accepted from the MATERIEL producer).

A recommended starting point for MATERIEL-TYPE data is the Baseline Resource Items List (BRIL) used in the CSSCS of the ABCS. The CSSCS provides the Logistics link between the Global Combat Service Support System – Army (GCSS-A) of the Standard Army Information Systems (STAMIS) and the ABCS. Therefore, it represents the minimal list of MATERIEL-TYPE data that is required by the operational user. Per the CSSCS web site[41], "The BRIL…is the master catalog table of supply items and personnel Military Occupational Specialties (MOS) available for tracking by CSSCS. The BRIL specifies the technical identifiers, class of supply, personnel information and the nomenclatures for those items (equipment, munitions, fuel, supplies and grade-MOS) that may be tracked. It is the key table within the CSSCS application."

The structure of MATERIEL-TYPE data has to be determined. There are (at least) four applicable data models (DMs) from which the structures contained in the Materiel server can be derived: the JCDB DM, the AICDM, the DoD's Core Logistics Data Model (CLDM),[42], and the DM from the Army's Logistics Integrated Database (LIDB).[43] Because each was designed based upon a different set of criteria, each has its own nuances. However, the CLDM, LIDB, and AICDM each have an entity called MATERIEL-ITEM-SUPPLY to maintain MATERIEL-TYPEs that are included in the DLIS FEDLOG catalog.[44] All four data models include an entity called MATERIEL-ITEM to store any MATERIEL-TYPE data regardless of whether it is contained in the DLIS FEDLOG. In the JCDB, a MATERIEL-ITEM is linked to a cataloged MATERIEL-ITEM via the BRIL that can be linked to a cataloged MATERIEL-TYPE in the GCSS-A. Therefore, a good starting point is to select a minimal set of attributes from the MATERIEL-ITEM and MATERIEL-ITEM-SUPPLY entities of the four mentioned data models. However, since the initial emphasis is

---

40 MATERIEL-TYPE is synonymous with MATERIEL-ITEM as defined in the DDDS.

41 From http://www.lee.army.mil/csscs/bril/pass_request.htm.

42 Corporate Logistics Data Model; see http://www.dlmso.hq.dla.mil/Data/Default.htm.

43 Logistics Integrated Database; see http://weblog.logsa.army.mil/index.shtml.

44 See http://www.dlis.dla.mil/govord.htm for information.

to empower the operational warfighter, the seven attributes from the JCDB DM's MATERIEL-ITEM entity must be appropriately mapped.[45] However, if all the databases use enterprise keys, then mapping between databases will be a relatively simple task.

Recommendation 4: A Materiel server database should be developed to serve as the authoritative source of stationary data for the Materiel domain, and its data structures should be derived from the best features of the JCDB, AICDM, CLDM, and LIDB data models to facilitate direct access by Army battlefield C2 and Logistics systems. Initially, the Materiel server prototype should be developed to validate the concept that includes only MATERIEL-TYPE data (e.g., DLA catalogued classes).

Eventually, MATERIEL-TYPE data that is not in the DLIS FEDLOG (i.e., do not have national stock numbers) will be desired. Examples are generic categories of MATERIEL-TYPES, such as armored vehicles, fixed winged aircraft, and amphibious ship classes. A particular model, or classification tree, about the nature and relations of beings is called a taxonomy.[46] The current MATERIEL-TYPE or taxonomy from the JCDB DM is limited (See Figure 3.2.2-1)

MATERIEL-TYPE	MATERIEL-TYPE-ASSOCIATION
- CONSUMABLE-MATERIEL-TYPE	
- - MUNITION	
- EQUIPMENT-TYPE	EQUIPMENT-TYPE-CONSUMPTION
- - POWER-GENERATION-UNIT	
- - NETWORK-DEVICE	
- - - NETWORK-BRIDGE	
- - - NETWORK-CABLE	
- - - NETWORK-GATEWAY	
- - - NETWORK-HUB	
- - - NETWORK-RADIO	
- - - NETWORK-ROUTER	
- - - WORKSTATION	
- - SENSOR-TYPE	
- - SHELTER	
- - SOFTWARE-APPLICATION	
- - VEHICLE-TYPE	
- - - LAND-VEHICLE-TYPE	
- - - WATER-VEHICLE-TYPE	
- - - AIRCRAFT-TYPE	
- - WEAPON-TYPE	

Figure 3.2.2-1. JCDB Data Model MATERIEL-TYPE Taxonomy

Although the current system includes some levels of hierarchical decomposition, a broader and more rigorous ontology is required to provide a wider range and higher levels of abstractions for

---

45 These attributes are: materiel-item name, technical identifier, descriptive text, symbol code, category code, supply class, and BRIL track identifier code.

46 *Taxonomy*: 1 : the study of the general principles of scientific classification; See http://www.m-w.com/cgi-bin/dictionary.

battlefield systems. The development of this capability is a major undertaking that is insidiously arduous because there is no single "correct" taxonomy capable of supporting all the different ways in which users categorize their data. Although a more complete ontology can be developed using the DLIS FEDLOG information system as a starting point, a realistic solution must include distributing the task across functional areas. This allows domain experts to participate in the ontology definition process thus providing a more precise categorization that closely corresponds to the real items. For example, one approach would be to allow each of the Army IMMCs to develop a taxonomy for those items under their purview. However, this approach would require significant inter-service cooperation for items with no obvious service proponent.

Recommendation 5: A process should be developed to build a taxonomy for MATERIEL-TYPE data that includes Logistics experts and data modelers followed by the development of a MATERIEL-TYPE taxonomy appropriate for Army and joint Logistics databases.

For a Materiel server to be successful, it must be continuously maintained and updated. To accomplish this, a definitive source within the Army must be identified that will maintain the information, and who preferably already does so. During this study, contacts were made with several organizations; they included: the Defense Logistics Agency data administration team (the DLA Component Data Administrator (CDAd) and the DoD Logistics Functional Data Administrator (FDAd)), the Army Materiel Command (AMC) Corporate Information Office, the Lead AMC Integration Support Office (LAISO), and the AMC Logistics Support Agency (LOGSA). An excellent source of materiel related information is in the Logistics Integrated Database (LIDB) maintained by LOGSA[47]. It has a comprehensive capability to combine data from several sources to produce a wide variety of reports.

Recommendation 6: The LOGSA LIDB should be evaluated as a candidate information source for a materiel server, and if discovered to be feasible, LOGSA should be resourced to develop a prototype Materiel server that contains MATERIEL-TYPE data.

A large benefit emerges from the synergistic effects gained by integrating the AOS data with that from the Materiel server. If accomplished, a collective set of highly detailed data about ORGANIZATIONs and their authorized equipment, down to the individual billet level, can be accessed and downloaded into Army and joint command and control and Logistics battlefield systems. This allows every system using the stationary data to cache its own local copy of the stationary data so that it doesn't have to be exchanged during daily operations.

Recommendation 7: A multi-disciplinary team should be assembled to establish a direct interface between the AOS and the Materiel server. This team should develop procedures to implement rigorous change management procedures between the stationary data servers so that fielded C2 and Logistics systems can fully exploit and rely upon the information provided by these stationary data servers.

---

47 LOGSA URL: http://www.logsa.army.mil.

3.5 Logistics and C2 System Integration – Enterprise Key Implementation

In any information system, a critical feature is the ability to link together disparate pieces of data and information via relationships. One way to greatly facilitate this task is to provide a common technique for identifying the pieces of information so that they can be conveniently referenced using a common format. This is the objective of enterprise keys. If all data is referenced using a common format, then one can effortlessly "plug and play" any data and information.

Recommendation 8: The Materiel server should utilize an enterprise key scheme based upon DOOs.

EIDs greatly simplify the process of integrating C2 Logistics stationary data. If LOGSA were selected to maintain the materiel server, then an initial ORG-ID could be assigned to LOGSA to establish an EID Server. In other words, if a single ORGANIZATION is created (i.e., the root ORGANIZATION called LOGSA), then a single EID server can be implemented that can assign over 4.3 billion EIDs, which far exceeds the few million EIDs required to uniquely identify the current set of MATERIEL-TYPEs. Ultimately, the DOO could be refined to include billet level resolution, but this is not required to produce enough EIDs for a prototype materiel server. An EID server is simply a piece of software that passes out unique numbers. It is recommended that the format of the EID be 64-bit binary identifier with the first 32 bits being the ORG-ID of the ORGANIZATION that sponsors the server and the second 32 bits being a unique value of the server's choice.

Once a Materiel server is completed, the AOS can be enhanced to include information about authorized quantities of the MATERIEL-TYPEs. A direct link can be made between the servers so that information can be continuously exchanged using a common data schema. However, the true payoff is the capability to download accurately maintained reference and stationary Logistics data to the tactical units in the field. Thus, the digital TOE/MTOE becomes a reality. The tactical units are then free to re-link the reference and stationary data as required. This can be accomplished efficiently via the EID system and the operations order and situational awareness processes already in place. But because each system has a common set of reference and stationary Logistics data pre-loaded into its database, the coordination required to disseminate the changes is greatly reduced.

APPENDIX A

ORGANIZATIONS AND INDIVIDUALS CONTRIBUTING TO THE PROJECT

APPENDIX A – ORGANIZATIONS AND INDIVIDUALS CONTRIBUTING TO THE PROJECT

Authors and Contributors

Michael Boller, LTC	U.S. Army TRADOC Program Integration Office – Army Battle Command Systems (TPIO-ABCS)
Samuel Chamberlain, Ph.D	U.S. Army Research Laboratory
Bruce Haberkamp	ODISC4
Richard Helfman, Ph.D	U.S. Army Research Laboratory
Cynthia Howell	AmerInd, Inc.
Henry Lavender	AmerInd, Inc.
Francisco Loaiza, Ph.D.	Institute for Defense Analyses

Sponsor

Deputy Under Secretary of the Army for Operations Research (DUSA(OR))	Mr. Walt Hollis

Stakeholder

Office of Deputy Chief of Staff for Logistics (ODCSLOG)	BG H.A. Curry

COTRs

Office of the Director of Information Systems for Command & Control, Communications, and Computers (ODISC4)	Mr. Bruce Haberkamp
Logistics Integration Agency (LIA)	Mr. Michael Blackman

APPENDIX B

COORDINATION EFFORTS

APPENDIX B – COORDINATION EFFORTS

Defense Logistic Agency (DLA).
POC: Dr. Stephen Broussard, DLA Component Data Administrator and Ms. Marge Larson, DoD Logistics Data Administrator.
Meeting on Thu, 25 May 2000.

AMC Corporate Information Office (CIO) Office.
POC: Mr. Danny Shearer, Chief, Core Technology and Assessment Division.
Meeting on Thu, 23 Mar 2000.

Army Materiel Command (AMC) Logistic Support Activity (LOGSA).
POC: Mr. John Peer, Manager for the Logistic Integrated Database (LIDB).
VTC on Wed, 14 June 2000 and meeting on Fri, 7 July 2000.

Lead AMC Integration Support Office (LAISO).
POC: Ms. Kathy Wachs and Mr. John Mays.
VTC on Mon, 15 May 2000.

In addition, the following personnel participated in the Start Work Meeting held on February 22, 2000:

Ms. Suzanne Acar	DISA
Ms. Pat Brislin	AmerInd, Inc.
Mr. Paul Gross	RDD USAFMSA
Me. Richard Hayes	USAFMSA
Mr. Leo Holly	SRA
COL (Ret) Dave Measels	SAIC
Mr. Richard Polacheck	DISA
Mr. James Raze	SRA
COL Michael Stine	HQDA, DCSLOG

APPENDIX C

RELATIONSHIP TO OTHER ARMY AND DOD INITIATIVES

APPENDIX C – RELATIONSHIP TO OTHER ARMY AND DOD INITIATIVES

In CY 2000, TRANSCOM began exploring better ways of managing all reference data that affect Logistics to improve configuration control and ensure consistent synchronization of all systems using this data. As discussed in this paper, Enterprise Identifiers (EIDs) provide a way to implement solutions to this problem.

The Army Enterprise Architectures process has already adopted an approach, which assigns "ownership" of architecture data to specific organizations, and is in the process of recommending a key management scheme that could readily adopt EIDs.

USAFMSA prototyped an Army Organization Server (AOS) using ORG-IDs. As a result, the Force Management System (FMS) Operational Requirements Document (ORD) includes a "must have" requirement for an AOS.

APPENDIX D

A COMPUTER DATABASE AND LOGICAL DATA MODEL PRIMER

Appendix D - A Computer Database and Logical Data Model Primer

Introduction

The purpose of this appendix is to attempt to present some basic information in non-technical terms about how computer relational database systems work and the purpose and role of logical data models in the design of interoperable information systems. The intent is to help functional managers and leaders, who may not have technical training in these areas, to better understand the implications of introducing the use of enterprise keys in DoD information systems and how they contribute to information interoperability.

D.1 Computer Databases

One of the things that can be very confusing about the study of how databases work and the role of data models is that there are many synonyms for some of the key terms and concepts that will be introduced. For example, later we will introduce and define the term, *data element*, which also has synonyms of *attribute*, *field*, and *column name*. The use of these synonyms, however, is generally associated with specific aspects of database design and data modeling. When referring to a data element in a particular context, such as a logical model of the structure of data in a database, it is called an *attribute*; or if describing the structure of how data is actually stored in a physical medium it is commonly called a *column name*; and when a data element is formatted for display of its contents in a presentation device it is usually called a *field*. As we make our way through this discussion, we will boldface and italicize the first introduction and definition of these terms to alert the reader to those things that are important to the development of an understanding of how databases work and the role data models play in the design of databases.

We will begin the discussion by establishing what we mean by the term ***database***. In the most fundamental sense, a database can be defined as any repository of data about any idea, abstract concept, physical object, subject, or any "thing" about which we might want to collect and preserve data over time for some purpose. We will use the term ***entity*** for all the different kinds of "things" for which we might want to collect data.

A database can exist in many different mediums. For example, a library of books is a kind of database; so is a filing cabinet full of folders containing information about "stuff" that we wish to keep for future reference; or even a three ring loose-leaf notebook. But a simple collection of data is of marginal utility to us if we cannot find the particular item of data we might be looking for, or if we are unable to interpret the data after we have found it. We need something else. We need something that describes to us what kind of data is contained in the database and where it is located. That is exactly what a 3.5" card catalog file does for a library, or a file plan for a file cabinet of folders, or the index page of our loose-leaf notebook. They "describe" certain properties or attributes about the data that is contained in the library, filing cabinet, and loose-leaf notebook, that permit us to find and interpret the data. So a more accurate and complete description, or definition, of a database would include the phrase, "any *self-describing* collection of data…". The technical term for this kind of "self-describing" data is metadata[48]. ***Metadata*** is frequently defined very simply as "data about data", and it is stored in a special part of the database called the Data Dictionary. More will be said about Data Dictionaries and metadata later.

Book libraries, filing cabinets, and loose-leaf notebooks are examples of "physical" or "manual" databases that we can touch and feel. These kinds of databases might be acceptable for our needs if they are not very large, and if we do not have a requirement to modify or edit the data very often after it has been inserted into the database. However, these kinds of databases quickly become too difficult and impractical to use when the amount of data we wish to store becomes very large, as on the order of millions and even billions of individual items of data. The task of attempting to manually insert, find, retrieve, update and delete selected items of data in acceptable time frames, and simultaneously maintain the accuracy and integrity of the data items in such large databases is in practical terms an impossible assignment. The difficulty is compounded when individual items of data may be duplicated in many locations throughout the database through the existence of redundant data elements in the database. (More on this to follow). This is where modern computer technology excels with database systems that are able to store, manage/process, and retrieve very, very large amounts of data with lightning speed and very high levels of accuracy and validity - provided they are properly designed. The remainder of this section of this appendix will focus on the structure and characteristics of properly designed computer databases.

D.1.1 Data Representation in Computer Databases

- To start our discussion of how data is represented in computer databases we will begin by defining what we mean by data and information to insure a common frame of reference for understanding[49].

  - **Data.** A representation of facts, concepts, or instructions in a formalized manner suitable for communication, interpretation, or processing by humans or by automatic means. For example, the symbol, "5", and the string of symbols, "five", are both English language representations of the same concept which by common convention we all interpret in the same manner. In other words, we all agree to attach the same meaning to the symbol "5" and the string of symbols, "five".

  - ***Information***. Any collection or communication of data related to one another in a specified structure such that meaning, knowledge, or intelligence is conveyed to a receiver of the data. The item of data represented by the string of numbers, 578629816, by itself means nothing. It could represent almost anything--a part number, or a bank account number, or perhaps a population statistic. But when we associate that number in a specific context with other items of data, it becomes information. For example, "The Social Security

[48] See DoD 8320.1-M-1, DoD Data Standardization Procedures, April 1998.

[49] These definitions are paraphrases of similar definitions found in numerous source texts on database theory and design. One relevant specific source is DoD 8320.1-M-1.

Number of employee 765 in Department 8 is 578629816”. (Data items underlined).

The first thing to remember about data stored in a computer is that as far as the computer is concerned, the data has no meaning other than a collection of electronic bits whose patterns represent symbols to which one can ascribe meaning. But, to be able to instruct the computer on how to process and manage these symbolic bit patterns, one must precisely define for the computer (as well as ourselves) the semantics, or meaning, and the syntax, or structure, of these bit patterns. For example, if one wanted to store data about "age" as it relates to individuals, one would have to define for the computer the number of symbols that will represent an "age" (perhaps three symbols to allow for age values greater than 99); the kind or type of symbols, i.e., whether numeric symbols that can be used in a mathematical calculation, or characters that are treated just like letters of the alphabet; a precise definition of what kind of "age" the symbols represent (age since birth, or age of majority, or legal drinking age, etc.); and finally, a name for this group of up to three symbols to help us distinguish it from other groups of symbols that are in the database. When we have accomplished this, we have created something called a ***data element***. The data that describes the characteristics, or properties, of a data element, as well as the data it contains, such as its name, definition, maximum number of symbols, or field size, its data type, etc., are known as the ***data element metadata***. The actual data, i.e., the group of one or more symbols that is contained within a data element, is referred to as a ***data item.*** The meaning we ascribe to a data item is commonly referred to as the ***data item value***. We will formally define these terms as[50]:

- ***Data Element***. A unit of data whose representation of a fact or concept within a given context is considered indivisible, and for which the definition, representation, and permissible values are specified by a set of metadata attributes.

- ***Data Item.*** A group of one or more symbols or characters associated with an instance of a data element whose meaning is defined by the metadata that describes the data element. The meaning these symbols represent is called the data item value.

- ***Metadata.*** The information stored in data dictionaries, data models, and database schemas that define and describe the characteristics of data in a database and its representation in a computerized system, i.e., data about data.

In the definition of a data element there are two important concepts that deserve further elaboration. The first is the phrase "considered indivisible", and the second is the idea of "permissible values". Let's first examine the phrase "considered indivisible". This is a very important objective in the design of data elements. Exactly what does it mean? It means that the group of symbols represented by a data element should not be able to be parsed, or subdivided into sub-groups of symbols so as to represent different meaning about different things. If they

[50] Again, these definitions are a composite paraphrase of similar definitions found in numerous texts on database theory and design. See DoD 8320.1-M-1 for specific DoD approved versions.

could, then this would be an example of what is commonly referred to as an "intelligent" data element--something that should be avoided in a properly designed database.

Let's look at an example of an intelligent data element from the world of DoD Logistics called a National Stock Number (NSN). An NSN is the universal catalog number assigned to every type of materiel item that is used by DoD organizations. It is a ubiquitous data element that is used by almost every type of transaction in DoD Logistics operations. Some of the metadata that describes the NSN data element tells us that it is composed of 13 symbols that are character type symbols. This would be an example of the symbols (a data item) that represent a National Stock Number for a type of materiel item - 2915015613452. Figure D.1.1-1 is a graphic representing the apparent structure of this NSN data element.

Data Element	**National Stock Number (NSN)**												
13 Characters	1	2	3	4	5	6	7	8	9	10	11	12	13
Character Value	2	9	1	5	0	1	5	6	1	3	4	5	2

Figure D.1.1-1. Apparent Structure of the National Stock Number Data Element

If we did not know any better, we would think the purpose of these 13 numbers serve only to identify a particular type of a materiel item. However, a closer examination of the actual structure of the NSN data element in Figure D.1.1-1 would reveal that the symbols also represent additional meanings as depicted in Figure D.1.1-2.

Data Element	**National Stock Number (NSN)**												
13 Characters	1	2	3	4	5	6	7	8	9	10	11	12	13
Four Distinct Data Elements	FSG		FSC		NCB Code		National Item Identification Number (NIIN)						
Character Value	2	9	1	5	0	1	5	6	1	3	4	5	2

Figure D.1.1-2. True Structure of the National Stock Number Data Element

We can see from Figure D.1.1-2 that the structure of an NSN actually consists of a combination of four more basic data elements - a Federal Supply Group (FSG) represented by two digits, a class within the FSG, a Federal Supply Class (FSC) represented by two digits, a National Codification Bureau (NCB) Code, the code representing an organization authorized to propose creation of an NSN represented by two more digits, and finally a seven digit National Item Identification Number (NIIN), the number assigned to a materiel item. Does this mean that an NSN cannot be used as the identifier of a materiel item in a database? The answer is no. NSN may still be used as an identifier. However, what it does mean is that each individual part of an

NSN, the FSC, FSG, NCB Code, and NIIN, should be created as its own individual (atomic) data element in the database. The NSN identifier is then constructed by simply combining (concatenating) the four separate data elements into an NSN.

The reason it is important to avoid intelligent data items is that computer systems do not process very well queries for information that are represented by data items embedded in other data items. For example, consider querying a database for all the NSNs created by a particular organization represented by NCB Code = 01, and assume that the only place this code is represented in the database is embedded inside the NSN data element. Then, the necessary instructions to separate out the 5th and 6th digits of every NSN to find the specific two digit code of the organization would be unnecessarily complex and inefficient. However, if the database included a data element that only represented a National Codification Bureau Code, then the instructions are relatively simple as in, "For all NSN select NSN where NCB Code = 01". To reiterate - the data item contained by a correctly defined and structured data element should represent a single, indivisible fact about the entity that it describes. Data elements that are structured so as to represent a single, indivisible fact are commonly referred to as "atomic" data elements. One of the principal objectives of a correctly designed database is to insure that its data elements are structured as atomic data elements[51].

Now, let's examine the concept of "permissible values" in the definition of a data element. This means that for any data element that we may wish to create, we must also create a metadata attribute that defines an allowable range of data item values that any instance of a data element will be allowed to assume. This is called the ***domain*** of the data element. The domain of a data element is a reflection of the "kind" or type of data element we have created. Consider these examples of two different types of data elements--Person Last Name and Person Gender.

Let us now suppose that we define a metadata attribute for the size of Person Last Name to be 15 symbols in length. The question now is, what kind of symbols will we permit to represent a Person Last Name? The answer is in the domain definition that we will specify as "…the 26 ASCII alphabetical characters A through Z". What this now says is that any instance of a Person Last Name can consist of any combination of the 26 letters of the alphabet from one to fifteen characters in length. So how large is the domain of valid values for this data element that we have defined? Well, if one performs the combinatorial mathematics to find out how many possible ways the 26 letters of the alphabet can be combined first one at a time, then two at time, three at a time… to finally 15 at a time, one will discover that it is a very, very large number -- so large in fact, that for all practical purposes the domain can be considered unbounded[52]. This kind of a domain is called a general domain because no attempt is made to explicitly enumerate each of the permissible values in the domain. Instead, we just define the individual symbols that are permitted to represent each data item value in the domain, i.e., the 26 letters of the alphabet. Now let's look at another type of data element, Person Gender, a data element with a specific domain.

---

[51] An obvious reference to the atom as an indivisible constituent element of matter even though we know that even atoms can be split and divided through the process of nuclear fission.

[52] The actual number is given by the summation of 15 terms as: $(26_1)^1 + (26_2)^2 + (26_3)^3 + \ldots (26_{15})^{15}$.

The domain of Person Gender is intuitively obvious, as a person's gender is either male or female. However, we still have a choice in how to represent gender as a data item value. The data element length could be defined as six symbols long, so that we could specify the domain of valid values as "…'male' or 'female' ". This means that the data element, Person Gender, would only accept two data item values where one value would be the alphabetic string "male" and the other string would be "female". On the other hand, we could also choose to define the length of the data element to be one symbol long, and then define the domain of valid values to consist of the two characters "M" or "F". But then we would also have to further include metadata definitions for the symbols M and F, i.e., the fact that M = Male and F = Female. In either case, this kind of domain is an example of an explicit, or enumerated domain--so called because metadata defining the domain contains an explicit list of every valid data item value that the data element is allowed to assume.

Some time has been spent discussing the concepts of atomic data elements and data element domains because they are very important considerations in the design of a database. If we do a good job of creating only atomic data elements in a database and accurately and precisely defines their domains, then we can construct meaningful rules for monitoring and managing the consistency and integrity of the data items in the database, which will ultimately affect the quality of the data and our confidence in its accuracy.

A data element can be thought of as a "data container" that contains the data item about the "thing", or entity, which we wish to store. An entity may have one or more distinct data elements. Whenever we populate the data elements of an entity, we create an instance of that entity. The set of values stored in each of the data elements for a specific instance of a given entity is called a *record*. An example commonly used to visualize this concept is to think about a bank of post office boxes in a post office where a post office box is analogous to an instance of a data element and a bank of post office boxes is analogous to a database. Figure D.1.1-3 depicts this analogy with an array of P.O. Boxes that have characters of the alphabet as their "names". The "address" of a box in the array would be the intersection of the row and column in the array of boxes. If we wanted to place a piece of mail in Box F, or find out whether it was empty or not, we would have to know the "address" of Box F—in this case, Row 3, Col 4. Conversely, if we only knew the value corresponding to the name of a P.O Box, we could query the records to find its "address", and then instruct the automated delivery system to put a piece of mail therein. Notice that in Figure D.1.1-3 there are three boxes with no names. Physically, they do have addresses within the array of containers, but they are as yet unnamed. These may represent "unused boxes", which in accordance with the "business rules" of our imaginary Post Office, will receive a "name" only when we have a new user to sign up for a box. In databases it is customary not to create empty records, i.e., instances of an entity for which there are no values other than the "key" that uniquely points to that record. Furthermore, it is of extreme importance to notice that how and when records are created constitute part of the overall database design. These so-called business rules form part of the overall meaning of the data and should be explicitly documented to ensure data integrity and consistency. These concepts will be developed further as we progress through this discussion.

	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6
Row 1	Box A	Box F	Box U	Box D	Box G	Box I
Row 2	Box V	Box L	Box N	Box O	Box K	Box Q
Row 3	Box E	Box __	Box B	Box F	Box D	Box H
Row 4	Box M	Box P	Box Z	Box Y	Box T	Box __
Row 5	Box S	Box X	Box J	Box __	Box C	Box R

Figure D.1.1-3. Post Office Box Analogy for a Database

D.1.2 Organization of Data in Computer Databases

Data elements in a computer database system must be organized in some sort of coherent internal structure that the computer can recognize and understand for three principal reasons:

- The first reason is so that the system may be able to navigate quickly and easily through the database to find any piece of data or data item in response to some instruction that one may give to perform some operation on the data item.

- The second reason is to ensure that when instructions are given to the database to perform some operation on a data item that modifies the data item value in some way, that the modification will occur in every location in the database where that particular data item may be represented.

- And the third reason is to make it easier to develop clear and unambiguous software instructions for communicating our instructions to the database system.

With respect to the second reason of updating every representation of a data item in the database, this refers to two kinds of situations. The first is the case where there are duplicate, or "cloned", occurrences of a data element, all of which have the same name and contain the same data item. The second case is where two or more different data elements exist to represent the same identical data item--an example of redundant data elements. Both cases are undesirable as they often contribute to creating data conflicts and inaccuracies in the database. As a consequence, it should be no surprise that one of the major objectives in the design of computer databases is to eliminate the occurrences of cloned and/or redundant data elements in the database. There is one very important exception to this objective relating to the first case of cloned data elements, which is fundamental to the operation of special kinds of databases called relational databases which will be discussed later in this section. But to understand the exception, we must first understand the distinction between the two cases of cloned and redundant data elements. Using the post office box analogy, Figure D.1.2-1 and Figure D.1.2-2 illustrate the difference.

Figure D.1.2-1 represents the first case where we wish to store the name of an employee named "Sue Brown" as a data item in post office Box Q, and then cause Box Q to be duplicated (cloned) one or more times in the bank of post office boxes in the post office. Figure D.1.2-2 represents the second case where the data item "Sue Brown" is contained in totally different boxes, or data elements, such as Box Q, and Box S, and Box F, and so on.

	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6
Row 1	Box A	Box F	Box U	Box D	Box G	Box I
Row 2	Box V	Box L	Box N	Box O	Box K	**Box Q**
Row 3	Box E	**Box Q**	Box B	Box F	Box D	Box H
Row 4	Box M	Box P	Box Z	Box Y	Box T	Box __
Row 5	Box S	Box X	Box J	**Box Q**	Box C	Box R

Figure D.1.2-1. An Example of Duplicated (Cloned) Data Elements

	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6
Row 1	Box A	**Box F**	Box U	Box D	Box G	Box I
Row 2	Box V	Box L	Box N	Box O	Box K	**Box Q**
Row 3	Box E	Box	Box B	Box F	Box D	Box H
Row 4	Box M	Box P	Box Z	Box Y	Box T	Box
Row 5	**Box S**	Box X	Box J	Box	Box C	Box R

Figure D.1.2-2. An Example of Redundant Data Elements

In the first case in Figure D.1.2-1, when "Sue Brown" gets married and changes her name to "Sue Tyler", we can issue simple instructions to the database to change the name in Box Q to "Sue Tyler" and it will be changed in every box named Q. The database system only has to remember the multiple addresses that were assigned to each occurrence of Box Q, which in this case are Row 2, Col 6, and Row 3, Col 2, and Row 5, Col 4. This might be a marginally acceptable practice provided that we could be assured that the database would always remember all the "addresses" of every occurrence of Box Q in the database, so that when there is a need to make some modification to the data item represented by Box Q, the database would be able to find every occurrence of it. But, in general, it is a very inefficient and unnecessarily complicated way of managing the data in our database.

In the second case in Figure D.1.2-2, we would have to issue instructions to change each separately named box containing the name "Sue Brown" requiring the database to keep track of their names as well as their addresses in the database. And if any one or more of those boxes were also cloned, then the problem is greatly complicated. This is a much more complex situation than the first case, and in very large databases that might have numerous redundant data elements, this becomes an almost impossible task and almost guarantees compromising the consistency and integrity of the data in the database. Case 1 can be thought of as "a common data element with multiple addresses for the same data item", and Case 2 as "multiple data elements and multiple addresses for the same data item". As will be seen later, eliminating instances of cloned and redundant data elements (excluding the previously mentioned exception) will greatly reduce the complexity of the instructions that we must issue to the database when we wish to perform some operation on some data element in the database. It will also assure the consistency and integrity of the data in the database.

To illustrate the different ways data elements can be organized in a database, we will create a small sample database that might be found in a typical Department of Motor Vehicles to collect data about three entities, AUTOS, DRIVERS, and TRAFFIC VIOLATIONS. We will then create data elements for Auto Name, Auto Color, Auto Tag No, Driver Name, Driver Age, Driver License No, Violation Type, Violation Severity, Violation Penalty, and Violation Driver - a total of ten data elements.

(*Note to the reader: The problems and issues highlighted in the examples that follow may seem simplistic and relatively easy to manage. Just remember that our small example database of ten data elements serves only to demonstrate these issues. When applied to databases that may contain literally thousands of data elements about hundreds of entities containing millions of data items (or more), these problems can quickly become unmanageable.*)

D.1.2.1 Random Database Structure

One way to organize data elements would be to assign a unique identifier to each instance of a data element as it is created that would serve as the location "address" of the data item in the database. The Figure D.1.2.1-1 graphic represents this kind of organization where data items are recorded in the database in no particular structure, but rather on a "first come, first served basis. Visualize a data entry clerk in the Department of Motor Vehicles with a stack of unsorted data forms keying in data from each successive form in the stack in a random fashion. As each data item is keyed in, our database system assigns it to a location in the database.

Violation 3 Type ID Violation 3 Type	Driver 2 License No ID Driver 2 License No	Violation 3 Penalty ID Violation 3 Penalty
Auto 2 Name ID Auto 2 Name	Driver 3 Name ID Driver 3 Name	Violation 2 Driver Name ID Violation 2 Driver Name
Driver 2 Name ID Driver 2 Name	Violation 2 Severity ID Violation 2 Severity	Driver 2 Age ID Driver 2 Age
Violation 3 Driver Name ID Violation 3 Driver Name	Violation 1 Type ID Violation 1 Type	Auto 3 Color ID Auto 3 Color
Driver 3 License No ID Driver 3 License No	Auto 1 Name ID Auto 1 Name	Violation 2 Penalty ID Violation 2 Penalty
Auto 2 Tag No ID Auto 2 Tag No	Violation 2 Type ID Violation 2 Type	Violation 1 Severity ID Violation 1 Severity
Auto 3 Tag No ID Auto 3 Tag No	Auto 1 Tag No ID Auto 1Tag No	Driver 1 License No ID Driver 1 License No
Driver 3 Age ID Driver 3 Age	Auto 2 Color ID Auto 2 Color	Driver 1 Name ID Driver 1 Name
Auto 1 Color ID Auto 1 Color	Violation 1 Penalty ID Violation 1 Penalty	Violation 1 Driver Name ID Violation 1 Driver Name
Violation 3 Severity ID Violation 3 Severity	Driver 1 Age ID Driver 1 Age	Auto 3 Name ID Auto 3 Name

Figure D.1.2.1-1. Random Organization of Data Elements

The random assignment of data storage locations depicted in Figure D.1.2.1-1 does not produce a coherent data element storage structure. Notice that data about the names of drivers is actually duplicated in two data elements, Driver Name and Violation Driver Name -- an example of redundant data elements. This forces the database to try and keep track of every occurrence in the database of data elements that store the names of drivers to attempt to keep the data in the two data elements synchronized -- a very difficult task with this kind of a scheme. The scheme also fails to take advantage of the affinity that groups of data elements may have with each other as in the example of data elements that represent some fact about autos, or drivers, etc. In addition, the random organization of data element instances greatly complicates the task of designing the software required to instruct the computer on how to find a specific data element that one might wish to access for retrieval, update, or some other type of operation, such as a calculation or aggregation, and then to insure that all other redundant/cloned representations of the data element in the database are similarly affected. A better organization scheme might be to place these kinds of data elements into identifiable groups called records. Figure D.1.2.1-2 is an example of this kind of scheme.

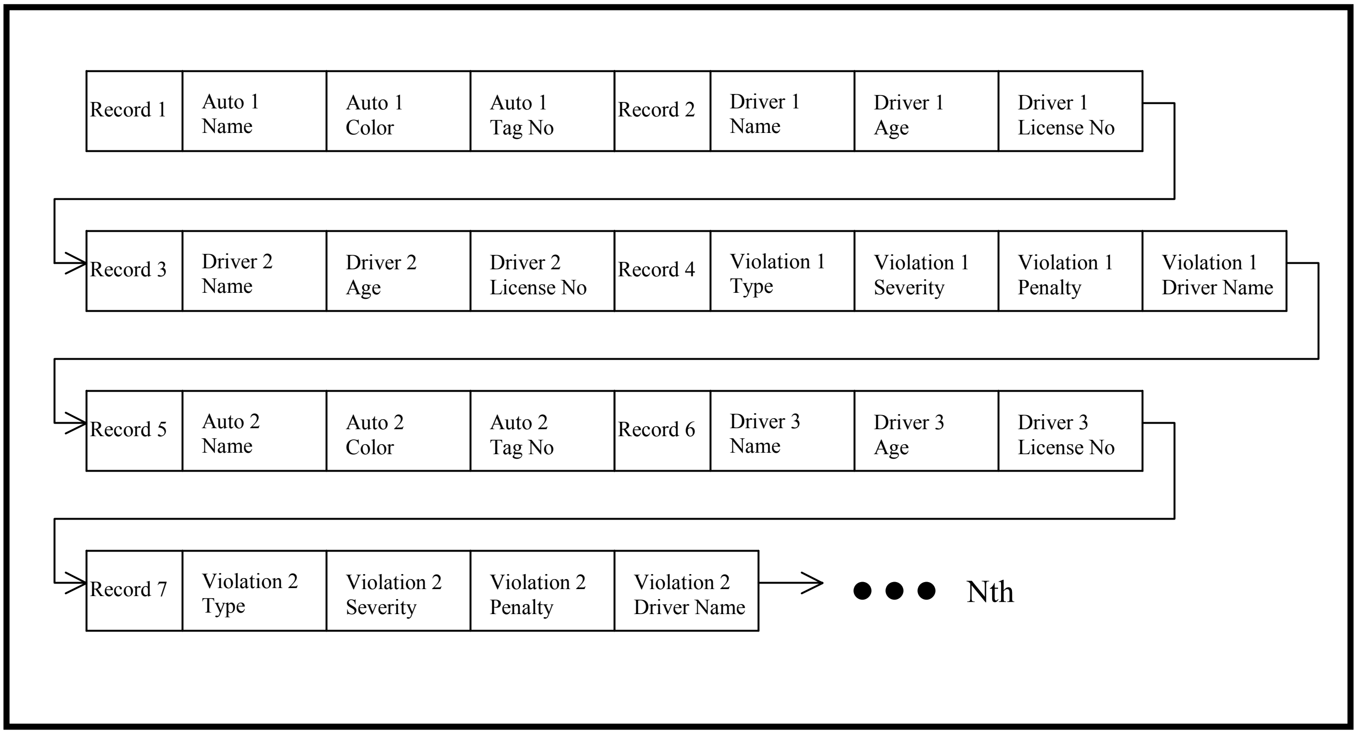


Figure D.1.2.1-2. Data Elements Organized into Records

D.1.2.2 Flat File Database Structure

Organizing data element into groups that relate to some common entity as depicted in Figure D.1.2.1-2 is clearly better than the random scheme in Figure D.1.2.1-1. The scheme in Figure D.1.2.1-2 is commonly known as a "flat file" database because it organizes the data elements that relate to the same instance of some thing we wish to collect data about -- like autos -- into a sequential series of groupings called records. A ***record*** can be defined as simply the ordered aggregation of data elements that all relate to a common ***instance*** of an entity. An instance is just a specific occurrence of an entity. So for the entity, Auto, the data elements containing the data items for each instance of an Auto that we enter into the database, namely, the Auto Name, Auto Color, and Auto Tag No, would all appear in the same record. This is the kind of database structure that was often used by database designers in the early days of computer technology circa the 1960s and 70s. However, notice that the problem of data element redundancy still exists in that a driver's name may still appear in more than one data element as in Driver Name and Violation Driver Name.

We must still identify each data element in each record with the instance of the entity that the data element relates to by including the instance identifier as part of the data element name as in Auto 1, Auto 2, Auto 3, and so on[53]. And finally, the occurrences of records that are related to instances of the same thing appear in a random order in the sequence of records. That is to say, data about Autos 1 and 2 are found in records 1 and 5, and data about Drivers 1, 2, and 3 are found in records 2, 3, and 6, and so on. It would improve our scheme considerably if we had a way of keeping or grouping all the records that relate to a specific entity into one coherent structure. And indeed there is a way to accomplish this by organizing records that relate to an entity into a single table[54]. The table then contains all the data that represents all the instances of the entity in the database. Figure D.1.2.1-3 illustrates this kind of table structure for organizing database entities and data elements into coherent structures.

AUTO

Auto No	Name	Color	Tag No
1	Mustang	Red	AYT-630
2	Camaro	Blue	ZKC-514
3	Lexus	Black	THF-108
4	Caprice	Blue	XTL-869
5	Mustang	Green	CDJ-276
↓	↓	↓	↓

DRIVER

Driver No	Name	Age	License No
1	John Doe	38	273627143
2	Jane Lee	25	734752081
3	Jim Jones	57	578431762
4	Bob White	19	362813825
5	Sue Tyler	43	296579264
6	Jane Lee	31	439698376
↓	↓	↓	↓

VIOLATION

Violation No	Type	Severity	Penalty	Driver Name
1	Speeding	55 in 35	$110	Sue Tyler
2	Parking	> 3 hrs	$3	Jane Lee
3	DUI	Felony	1yr +	Jim Jones
4	Speeding	95 in 65	$250	John Doe
5	Speeding	35 in 25	$5	Sue Tyler
↓	↓	↓	↓	↓

Figure D.1.2.1-3. A Table Structure for Organizing Data Elements in a Database

---

53 Knowledgeable readers will recognize that multiple occurrences of "instance identifiers" in each data element in a record could be eliminated by creation of a "header data element" following the Record No data element that identifies all subsequent data elements in the record as relating to a common instance of the entity we are collecting data about. So that for Autos, the records would read like this: |Record No|Auto No| Name|Color|Tag No|, where |Auto No| is the instance identifying header data element for the record.

54 There are also other ways besides tables for organizing records, such as a flat file scheme of grouping records into segments. But this appendix is concerned with relational databases that use the table structure.

D.1.2.3 Table Structure for Databases

In a database utilizing a table structure design, each entity that we may wish to collect data about is represented by a table in the database where the data elements are the columns of the table, the top row of column labels are the names of the data elements and each instance (or record of data elements as in the flat file structure) of the entity is represented as a row in the table. The cells of the table formed by the intersection of a row and a column contain the actual data, or data item, that is represented by the data element for that particular instance or row of the table. The location address of a cell (data item) in the table is the column (data element) name and the row identifier. No matter how large a database may get or how many tables it may contain, we need only to specify three things to locate any specific data item in the database--the table name or table identifier, the data element or column name, and a row identifier for a row in the table.

The problem of redundant data elements still remains as can be seen by the two data elements, Name, in the DRIVER table, and Driver Name, in the VIOLATION table. Both data elements exist to store the same kind of data items, i.e., the names of drivers. The Name data element in the DRIVER table exists because one of the facts or data items we want to know about a driver is the driver's name. Likewise, in the VIOLATION table that stores data about traffic violations, we are also interested in storing the names of the drivers that commit the violations. Therefore, we create a Driver Name data element in the VIOLATION table for this purpose. Thus, whenever a driver whose name is recorded in the DRIVER table has committed a violation, we must also enter his or her name in the VIOLATION table. The problem is that we have to remember to edit the name data in both tables anytime some change to the driver's name occurs--as in a marriage.

But even if we could do all of that correctly every time, we can still have a problem. Notice that there are two drivers in the DRIVER table with the identical name – "Jane Lee." Jane Lee is recorded in the VIOLATION table as having committed two violations. Did the same Jane Lee commit both violations? Or did each Jane Lee commit one violation, and if so, which violation did each Jane Lee commit?

Maintaining the data in our database would be much less complex and much easier if we had some method whereby we could record data in only one data element in one table and make it available to be shared by all other tables in the database that might require the data element as a descriptive characteristic. This would eliminate the need for redundant data elements and simplify the task of maintaining the data in our database.

D.1.2.4 Relational Database Structure

As it happens, such a mechanism for preventing the occurrence of redundant data elements does exist. The answer lies in creating relationships between the table structures of our databases that conform to the definition of a relational database as described by Dr E.F. Codd[55]. The

---

55 Relational databases conform to the definition of a relational database model that was an outgrowth of a seminal paper published in 1970 by E.F. Codd in which he applied the mathematical calculus of relational algebra to the problem of organizing large amounts of data for storage, manipulation, and retrieval in computer databases while maintaining the accuracy and integrity of the data.

relationships reflect how the entities and data elements relate to each other in the real world. In a relational database structure consisting of multiple tables, a ***relationship*** is a manifestation of the association that any table in a database has with another table. The entity, or table, from which the relationship originates is called the parent, and the target table of the relationship is called the child. The relationship represents constraints, or rules, that are imbedded in the database system that govern how a table in the database can share access to data items represented by data elements in other tables. They also specify the rules for how the data items will be modified in response to instructions that we may issue to the database. The relationships between tables in a relational database are often referred to as "business rules" because that is exactly what they are. The way the data relates to each other in the database is a reflection of the rules employed by the business or processes that are responsible for creating and using the data in the database. Figure D.1.2.4-1 depicts the relationships for our sample database when structured as a relational database.

AUTO

Auto No	Name	Color	Tag No
1	Mustang	Red	AYT-630
2	Camaro	Blue	ZKC-514
3	Lexus	Black	THF-108
4	Caprice	Blue	XTL-869
5	Mustang	Green	CDJ-276

DRIVER

Driver No	Name	Age	License No
1	John Doe	38	273627143
2	Jane Lee	25	734752081
3	Jim Jones	57	578431762
4	Bob White	19	362813825
5	Sue Tyler	43	296579264
6	Jane Lee	31	439698376

may be involved in

(0,N

(0,N

commits

VIOLATION

Violation No	Type	Severity	Penalty	Auto No	Driver No
1	Speeding	55 in 35	$110	3	5
2	Parking	> 3 hrs	$35	1	2
3	DUI	Felony	1yr + $10k	4	3
4	Speeding	95 in 65	$250	3	1
5	Speeding	35 in 25	$55	2	6

Figure D.1.2.4-1. Relational Database Structure

Figure D.1.2.4-1 looks very much like Figure D.1.2.1-3, but we have added to it some symbology that depicts how tables are related to one another. We also made a change to the name of one column in one of the tables and added another column to the same table. First, the symbols that depict the relationships have embedded meaning--constraints and rules--that will be reflected in the software code that translates instructions to the database to perform some operation on the data. For example, the arrows between the DRIVER and AUTO tables and the VIOLATION table indicate that the DRIVER and AUTO tables share some of their data with the VIOLATION table. This sharing occurs because the Department of Motor Vehicles, the "owner" of our database, has a business rule that says that every violation that is entered into the VIOLATION table must be associated with the driver that committed the violation and the auto that was involved. (The kind of arrows, as in solid or dashed, single or double headed, etc, and the "(0,N)" notation also convey other meanings associated with the relationship. These will be covered in more detail in the discussion on logical data models. But for now, all one needs to know is that they indicate a data sharing relationship). Second, notice that the name of the column of the VIOLATION table for the data element that the DRIVER table shares with the VIOLATION Table has been changed from Name to Driver No. Why did this happen, and how does this change allow the VIOLATION table to keep track of the names of drivers that commit violations?

To understand how relational database tables share access to data held within different tables in the database, we have to introduce a new term, called a "***primary key***". (A key is an identifier used to identify records in a table of which a primary key is just one of a number of types of keys that will be discussed in greater detail in the section on logical data modeling). For now, recall that we said that for a database that uses a table structure design that only three things are required to identify any data item in a table -- the table name or identifier, the column (data element) name, and a row identifier for a specific row in the table. Primary keys are nothing more than "special" kinds of data elements whose data item values in each row of the table are guaranteed to be unique for every row in the table, thus constituting row identifiers. Therefore, any specific value of a primary key will always reference only one specific record or row in the table -- the same row every time it is used. This allows us to be certain that when we are querying the database for information on violations and the names of the drivers that committed them, if we have constructed our queries using primary keys, then our system will know to look in the correct row of the DRIVER table for the name of the driver associated with a particular violation. Why not just refer to and use directly the Name data element in the DRIVER table instead of the seemingly roundabout reference to the Driver Name through the Driver No? The answer, of course, is that we cannot guarantee the names of drivers to be unique. (Just look in any phone book and count the number of people named John Smith.)

In our small database there are two people with the name Jane Lee -- one is 25 years old and the other is 31. If our VIOLATION table includes the Name data element from the DRIVER table as in Figure D.1.2.1-3, and we query the VIOLATION table to tell us the driver's license number of Jane Lee, how does it know which "Jane Lee" record (row) in the DRIVER table to access -- the 25 year old Jane Lee from row 2, or the 31 year old Jane Lee from row 6? The answer is it does not. However, if we instead share the DRIVER table data element, Driver No, with the VIOLATION table as in Figure D.1.2.4-1**,** and ask the same question for Driver No 2, our system will unerringly return the driver's license number of the 25 year old Jane Lee from the row of the DRIVER table identified by the Driver No primary key data element data item whose value is 2. This happens because we have designed our DRIVER table so that every instance of a driver in the DRIVER table is guaranteed to have a unique Driver No that is never repeated in the Driver No data element - **a primary key**.

With the exception of primary key data elements, a correctly designed relational database should never have to create a redundant data element in a database in order to share data items. There need only be one data element for every defined type of data item that we may need to represent in a relational database[56]. The example we have used is the names of drivers in our database. We record those names in just one place (data element) in the DRIVER table of the database. When we wish to make the data in the DRIVER table available to other tables in the database, we create a relationship to a target (or child) table which causes the primary key of the DRIVER table to be copied, or cloned, to the child table. When this occurs, the cloned primary key data element(s) in the child table is called a **foreign key (FK)**—meaning that the data element is "foreign" to the child table. Now when a change is made to a driver name in the database, it only has to be changed in one place in the DRIVER table and that change is instantly available to every other table in the database that has a relationship which is manifested as a foreign key pointing to a record (row) in the DRIVER table. (In fact, in some database texts the presence of a primary key from one table in another table as a foreign key is referred to as a "pointer" that points back to a record in the table of its origin).

This method of sharing data is the same as "cloning" data elements, which we previously said was to be avoided as a generally undesirable practice. However, this is the exception to that rule that we referred to as being acceptable because we are only cloning primary key data elements which are identifying data elements. The reason cloning data elements is generally a bad idea is because of the problem we have in making sure that when a change is made to the data, that the change gets implemented in every copy of the cloned data element, thus preserving the consistency and integrity of the data in the database. But we should not have to worry about this problem for primary key data elements because one of the properties of a primary key is that its values are stable and should never change once created. This will be discussed again in greater detail in the discussion of data models.

> *Database designers employ a well-defined methodology called "normalization" that uses a structured analysis process of examining entities (tables) and data elements (columns in tables) during the design of a database for the purpose of eliminating redundant occurrences of non-primary key data elements in a database. The details of this methodology are not discussed in this appendix. However, for interested readers, the methodology may be found in almost any database design textbook such as Designing Quality Databases with IDEF1X Information Models, Thomas A. Bruce, Dorset House Publishing, New York, NY, 1992. Suffice it to say here that a reference to a database that is said to have been normalized, means that the normalization methodology has been performed on the database to remove redundant occurrences of non-primary key data elements.*

By employing relationships that conform to the rules of our business or enterprise to permit the tables in our database to share access to data elements among themselves through the device of

[56] There are occasions where designers may wish to create instances of redundant data elements. Relational database theory does not prevent this. Normally these cases are related to improving database performance and generally involve data elements whose data item values tend to remain stable and unchanging over time. Examples of these kinds of data elements would be those resident in reference code tables such as two digit state codes or postal zip codes.

sharing the primary keys of tables, we have solved the problem presented by redundant data elements. Of course, we now have to be very careful that we fully understand all the business/process rules of our enterprise to be sure that we correctly define and create the relationships that are the implementations of those rules in our relational database system. But this is a doable task and should always be a central requirement of any database design project. Databases do not exist in a vacuum for their own sake. They exist to hold the data that is deemed necessary to the conduct of some purpose, activity or process. It is the rules that govern the execution of the functional activity or process that dictate the rules that must be employed in managing the data in the database. Serious and thorough functional activity/process analysis should always be a requirement when designing a database. This idea will also be discussed in greater detail in the discussion on data models.

D.1.2.5 Data Integrity in Databases

Numerous references have been made to maintaining the "consistency and integrity" of the data in a database to enhance the accuracy of the data item values in a database. It is important for the reader to have a basic understanding of what it means to make a reference to database consistency and integrity and how this contributes to the accuracy of data in a database. The first thing that one must understand is that with respect to the data item values in a database, it is entirely possible for data item values to be inaccurate and still be said to have integrity. This is because the concept of data integrity only means that when data item values are entered into a database, their existence throughout their life in the database is at all times logically consistent. In other words, if the constraints contained within the database for manipulating data complies with the enterprise business rules governing the manipulation of the data, then the database can be said to exhibit data integrity. These constraints can operate at different levels in a database. For example, there are database constraints that operate to ensure that data element data item values are consistent with the domain of valid values defined for the data element. This is called ***domain integrity***. Similarly, there are constraints that operate to ensure that the rules for sharing data elements through relationships between tables are correctly enforced and this is called ***referential integrity***. The operation of rules designed to guarantee the uniqueness of instances of an entity, or the records (rows) of a database table is called ***entity integrity***. There are also other classifications of data integrity such as ***database integrity, transaction integrity, procedural integrity, declarative integrity***, etc. A complete discussion of these concepts is beyond the scope of this appendix. However, the subject area usually receives extensive treatment in most textbooks on designing databases. For the interested reader, one particularly good text on the subject is, *Designing Relational Database Systems, Rebecca M. Riordan, Microsoft Press, Redmond, WA, 1999.*

We might now ask how could it be that data may have integrity and still be inaccurate? With respect to domain integrity, it is very possible for one to enter an incorrect data item value into the database that in every respect complies with the definition of the domain of valid values for the data item. A simple example would be in the case of the earlier example described as Person Gender Code whose domain of valid values is limited to either "M" or "F" so that any other value would be rejected by the database at data entry time. Now suppose we were to make a mistake in entering a value for a female person and entered a code value of "M" for a female instead of "F" for female. The code value "M" is indeed a valid value in the domain of Person

Gender Code, and would be accepted by the database, but obviously it would be inaccurate for a female person. Yet, the data item value could rightly be said to display domain integrity. Similar examples could be developed with respect to how database referential integrity rules reflected in the relationships between tables operate to maintain consistent and correct data item values during insertion and deletion operations.

So why is it so important to strive for data integrity in the design of a database if the data can still be inaccurate? The answer is because there are many ways to develop inaccuracies in data item values beyond simple data entry errors -- often referred to as "fat fingering" in the data. A database that does not reflect data integrity by either not correctly executing the enterprise business rules and constraints for manipulating the data in the database, or by just failing to include all relevant business rules in its schema, will generate inaccuracies in the data item values. Thus, even though rules and constraints that operate to enforce data integrity in the database may not always be able to guarantee total data accuracy, they can ensure that subsequent to data entry further manipulation of the data will not introduce inaccuracies in the data. This is why it is so important to provide for data integrity in the design of a database. The bottom line is that providing for data integrity in database design is an important element of maintaining the accuracy of data in a database.

D.1.3 Database Management Systems (DBMS)

So far we have focused only on describing the key features of how data is organized for storage in a database and, in particular, a relational database. We defined the principal structural elements of a relational database such as data elements, tables, primary keys, and relationships, and their role in maintaining the integrity and accuracy of the data in the database. But a database by itself is just a repository of stored data item values that will allow us to access, add, delete, or update those data items in response to business requirements. But we cannot do any of those things unless we first have a way to communicate with the database to enter and delete data and perform other operations on the data as our needs require. To accomplish this, there is another part of the database system called the database engine that provides the communication interface between a user and the database.

The database engine contains the computer software code that processes our instructions to perform some operation on the data items stored in the database. It also includes the software program code that describes the design and organizational structure of our relational database, as well as including the embedded rules represented by the relationships between the tables in the database. These two parts of the database engine are usually called the Data Definition Language (DDL) which contain the coded instructions for creating the database structures in accordance with our design (commonly known as the database schema), and the Data Manipulation Language (DML) that accepts and translates our instructions, commonly known as "queries," to the database.

Finally, the third part of a database system is the interface through which the user communicates instructions and requests to the database engine. The interface may be nothing more than a command line processor where we enter coded DML instructions directly to the database engine, similar to how we might issue commands to a PC DOS or Unix-based operating system. It can also take the form of a graphical based interface where we can build our instructions to the database by "pointing and clicking" on selected options from a series of menus and icons, similar to how most of us communicate with our PCs today through a windows type interface to the operating system such as Microsoft Windows. The interface can also be a software business application system that includes data input and output functions for communicating with the database through the database engine as data is created, modified, and deleted in response to business events and transactions.

The term, database management system (DBMS), includes, at a minimum, the database engine and the database. In some cases, the interface can also be considered to be part of the DBMS. The distinction is not significant. What is important is to understand the roles of each part of the system. The graphic in Figure D.1.3-1 depicts the relationship between the three elements of a database system -- the database, the database engine and the user interface.

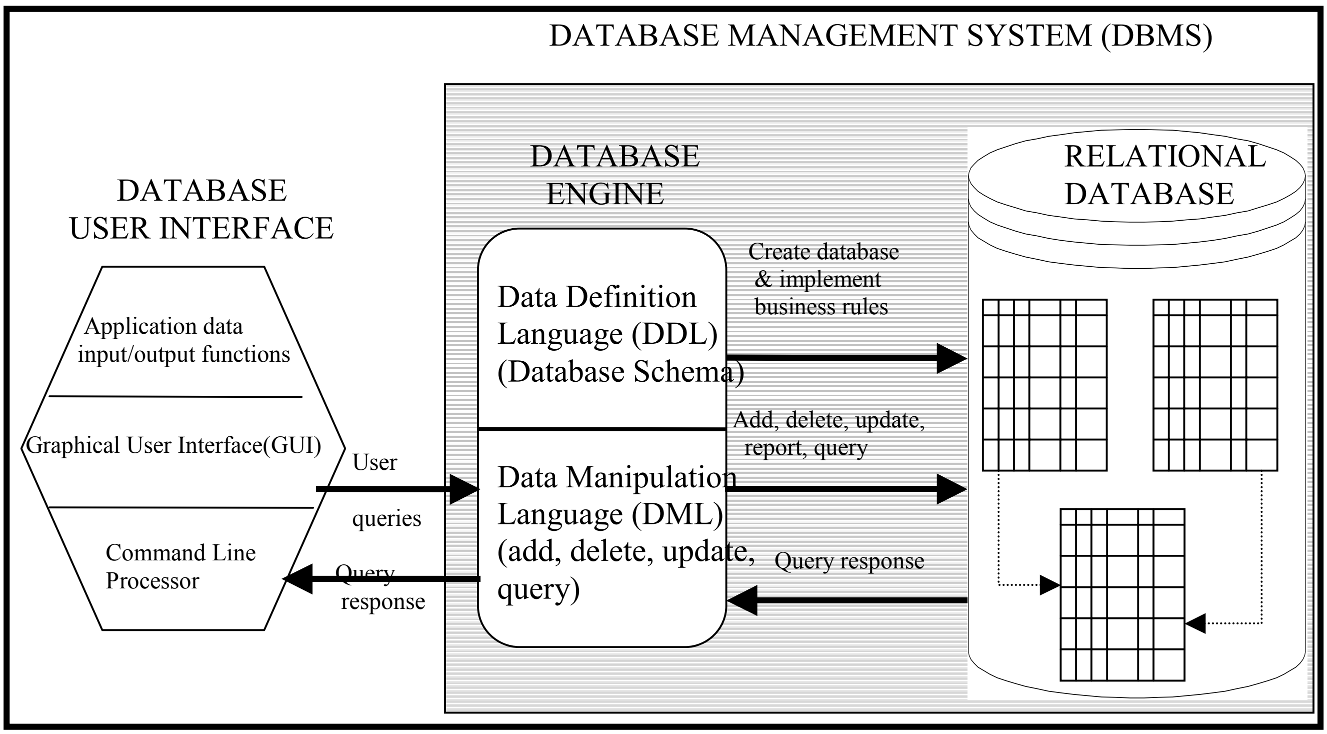


Figure D.1.3-1. Elements of a Relational Database System

ANSI Standard Structured Query Language (SQL) (pronounced "sequel") is the basis of most of the relational database engines employed by the DBMSs of the leading commercial relational databases such as Oracle, Informix, Sybase, Microsoft SQL, etc. Now if the DDL and DML of all these database engines reflected an exact implementation of ANSI SQL, then it wouldn't make any difference which database engine was used as part of the DBMS. However, this is not always the case. Most vendors of relational database systems modify ANSI SQL in their database engines in some way to include some kind of functionality not included in the ANSI SQL, which usually makes the various commercial database engines incompatible with each other. However, this does not pose any serious problem for database implementation. One of the more significant advantages of a database system organized as depicted in Figure D.1.3-1 is that the database and the data it contains exist independently of the database engine and user interface. For this to occur, the manipulation of the data in the database must be governed solely by the business rules of the enterprise that creates, uses, and manipulates the data and not by any particular version of a database engine or user interface, particularly if the interface is a business application system. This means that we can design and create a relational database and use any "brand" of application software and/or database engine to manage the database as long as the business rules contained in the database structure can be implemented by the database engine. The ability to create and maintain data and the database in which it resides separate from any particular application that may interface and operate on the data is another important design goal for properly designed databases. It creates for a business tremendous flexibility to modify its functional processes and software systems as needed in response to changes in the business environment without any fear of losing or damaging the database repository of data upon which the business depends.

D.1.4 Database Structures Summarized

In this discussion on the organization and structure of data in a database, we have endeavored to present a picture of how databases work. In our discussion, we have emphasized the relational database, because the relational model is the basis of almost all modern database management systems and is the most widely used throughout the DoD. The following is a review and summarization of the key points that have been made:

- A **database** is a self-describing collection, or repository, of facts about things and concepts, which are called **entities**, which we choose to store for some purpose.

- Individual facts, which are called **data items**, are represented in a database by groups of one or more symbols such as numerals and alphabetic characters that are called **data elements**.

- The structure and meaning of data elements are defined, and their characteristics described, by attributes called **metadata** which are stored in a separate part of the database called a Data Dictionary.

- Data elements whose data items represent facts about a common entity are aggregated into groups called **records**. A record represents a single **instance** of an entity, and a data item in a record represents an instance of the data element containing the data item.

- For a relational database, records that represent instances of a common entity are organized into **tables** where the entity instances, or records, represent the rows in the table and the columns of the table are represented by the data elements.

- A **primary key** is a special kind of data element whose data item value constitutes a unique identifier of an entity instance, or row, in the table representing an entity. To qualify as a primary key data element of a table, the value of every primary key data item within the table must be guaranteed to be unique.

- The location, or **address**, of any record in any table in the database is defined by the name, or identifier, of the table and the primary key of the record. Similarly, the address of any data item in the database is defined by the table name, data element name, and primary key of the row in which the data item resides.

- The integrity and consistency of data represented in the tables of a relational database is maintained through **relationships** between tables. These relationships are manifestations of the business rules of the real world purpose for creating the database that govern the creation, deletion, update, and manipulation of data records and data items in the database.

- The implementation of a relationship in a relational database is represented by **sharing** the primary key data element(s) of the parent table in the relationship with the child table in the relationship. The shared primary key data element(s) in the child table is called a **foreign key** in the child table.

- A **database system** is composed of a database, a **database engine**, and **a user interface**. The database engine includes the DDL which specifies the structure of the database entities, their data elements, and their relationships, and the DML which accepts user instructions and queries and executes them in the database consistent with the business rules defined by the DDL. The user interface transmits user instructions and queries to the database through the database engine and reports back the results of queries.

- A **relational database management system** (RDBMS) consists of a database engine and a relational database and sometimes the user interface.

D.2 Enterprise Business Models

Throughout Section 1 of this appendix we have made numerous references to "properly designed databases". Now we will begin to address how to achieve this goal of a “properly designed database”. As we will attempt to develop in the following discussion, the foundation of a good database design begins with an accurate data model, in particular, an accurate *logical* data model to differentiate it from other types of data models such as conceptual and physical data models. The differences will be made clear as we proceed through the discussion. But first, let us briefly consider what a model is in general, because a data model is not the only kind of model that we should be using in developing a design of our business database.

A model is nothing more than a representation or symbolic abstraction that specifies one view of some physical or conceptual object that exists in the real world. For example, the architectural blueprints of a high-rise building are kinds of models that together represent the design specifications of a real physical building. Each blueprint depicting different aspects of the design such as the structural design, or electrical schematic, or floor plan layout are each different views of the overall model of the building. Similarly, a mathematical formula is also a model of some abstract relationship such as the relationship between the three sides of any right triangle modeled by the familiar formula, $A^2 + B^2 = C^2$, known as the Pythagorean Theorem. A carefully drawn geometric diagram of a triangle that includes one interior angle of exactly 90 degrees represents another view of this relationship. Just as no competent architect would undertake to build a structure as complex as a high rise building without a complete model or set of blueprints representing the design specifications of the building, neither should a business information systems builder attempt to build a computer database system for a business without a complete set of design specifications that fully describe the business, activities, processes, and data[57]. Those specifications are represented in different kinds of models such as activity models, process models, and data models that represent different views of the business.

One important aspect of models that is often forgotten, or at least not given much consideration, is the fact that most models are developed in an iterative fashion. This is especially true of the various types of models that represent business activities and operations. This means that business models are usually developed in stages involving research and collection of data to develop preliminary models followed by analysis, more research and data collection, and refinement of the preliminary model. Then there is more analysis, and so on until a mature model evolves that is an accurate representation of the view of the business the model is designed to represent. The following are descriptions of three kinds of business models that are especially useful to database designers in the design of relational databases for business operations.

An **activity model** of a business depicts in a graphical diagram the functional processes that a business engages in and their relationship with each other. From a data point of view, each activity and sub-activity is described in terms of the data that is input to the activity, the controls that proscribe how the data may be manipulated by the activity, the mechanism(s) which are used to manipulate and transform the data, and finally the data that is output as a product or as the input to the next activity in the process. An important characteristic of a business activity model is that it does not represent how long any particular activity requires to complete its task(s). That is to say that if an activity is defined that exists to process a customer order, then the model only describes what is involved in processing the order, but not how long it might take, e.g., 30 minutes, 1 day, or one week. A shorthand way of describing this kind of model is to say, "An activity model describes ***what is done to business data and how it is manipulated*** without describing the structure or meaning of the data".

Process flow models depict the path or flow that the things that the business is concerned about takes through each functional activity/process whether these things are actual physical materiel objects, or the data and information used by the business. With respect to data, these kinds of models describe the data that is stored and maintained, the business organizations, people, projects and systems that interact with the data, and the transformations that occur to the data and how long they take.

---

[57] The analogy is so close that data modelers and database designers are often referred to as information system architects.

Data models describe the structure and meaning (semantics) of the data that is used by a business. A conceptual data model depicts at a relatively high level of abstraction the fundamental types of data that a business requires to function and how these data types which are called entities relate to each other. A conceptual data model usually does not contain any detail about descriptive characteristics of the entities, called attributes, or the specific detailed rules that govern their relationships. A logical data model represents an extension of the conceptual model in which all descriptive attributes of entities are defined and the specific rules governing relationships between entities are defined. Finally, a physical data model represents how the logical data model is actually implemented in a specific database design using a specific brand of software for the database engine. It is the logical data model that contains the detailed specifications that a database designer requires to actually build a physical database, and so that is what we will focus on for the remainder of this discussion. A shorthand way of describing a logical data model in contrast to activity and process models is to say, " A logical data model of business data describes ***what the data is, its structure, and its meaning*** without telling us anything about what is done to the data, nor how it is done".

So, where do activity and process models belong in the scheme of things with respect to designing a database? They don't tell us much about the structure and meaning of data, but they can tell us a lot about the business rules that govern the execution of business functions and how the data that is used by those functions is manipulated and transformed. These models can be important sources of information that can help us to correctly define the structure of the relationships between entities in our data model. Such relationships need to provide an accurate representation of the business rules that govern the management of data in the database.

The point of this discussion on models is to emphasize that while logical data models represent the foundation on which a database design can be created, other views of the business represented in activity and process models are also necessary for the designer to understand and reflect in the database design as appropriate. They are all important. But for the remainder of this discussion we will only deal with data models as the foundation of the database that is part of a larger business system.

D.2.1 Logical Data Models

The first thing to understand about a logical data model is that it does not--repeat--does not directly represent the design of an actual database that is used by some business activity or process. What it does represent is the structural and semantic specifications of the data that is required and used by the business and how that data relates to each other independent of the business activities and/or processes that use and process the data. A database designer then employs these specifications in designing an actual database. This is the central characteristic of a logical data model that it makes it so important in the design of any business system database including relational databases. If this concept is not completely clear, think about an example such as an insurance business.

The kind of data, i.e., the data entities and data items that an insurance business requires, is data about policy holders, policy beneficiaries, types of policies, premiums, claims, risk factors, benefits, etc. It doesn't matter if we are talking about an insurance business in 1900 or an insurance business today in 2000. The way the business is organized, meaning the functions and activities created to collect and process the data, may certainly change over time to accommodate changes in the business environment, such as changes in technology. But the underlying structure of the data upon which the business depends tends to remain stable. An insurance business in 1900 was organized to process and maintain the data important to the business on paper based forms, ledgers, and filing systems with armies of clerks to process and maintain the data. But an insurance company today relies on modern computer technology to maintain the data it needs to conduct business and is organized accordingly. However, the kind of data it needs has not changed. This concept is often summarized in database design textbooks with a statement that says something like, "Business functional activities and processes are subject to change at any time to accommodate to changes in the external business environment, but the underlying structure and relationships of the data elements required by the business tend to remain stable over time". What all of this really means, is that if our database design is indeed an accurate reflection of the structure of business data requirements, then we have created the ability and flexibility to modify business activities and processes as required without having to worry about changing the structure and content of the underlying business database. This is also what is meant by the goal of achieving the separation of business data from business applications in the database design. So now let's look at what constitutes a logical data model.

D.2.2 The Structure of Logical Data Models

A logical data model is comprised of two parts. The first part is a diagram that graphically and visually depicts the three principal data objects of a relational database -- entities, attributes, and relationships -- and the business rules governing the nature of each relationship. If any part of the professional discipline involved in designing databases could possibly be considered glamorous or "glitzy", it would be the art of creating accurate and elegant data model diagrams. The data model diagram is what most people think of when they think about data models. The diagram is important because it is a simple and easy method for communicating to a database developer the specifications of the structure and relationships of the data objects that are required by a business activity or process for the database developer to consider in the design of a database that will be implemented and managed with a particular brand of database management software.

The second part of a logical data model is the metadata dictionary that contains the specifications of all the data objects in the database. The tasks involved in building a metadata dictionary require disciplined business analysis, based on a thorough understanding of the "business rules" of the business activities/processes that will use the data, and careful attention to detail to be sure that all required metadata is developed and recorded for each data object in the database. The process is fairly straightforward and methodical--not as "glamorous" perhaps as creating the ER diagrams, but no less important. (Many database designers consider the metadata dictionary to be the most important part of a good logical data model).

Creating a logical data model diagram, on the other hand, is almost as much of an art as it is a rigorously defined process. That is because it is often possible to create different but equivalent versions of a diagram to represent the business data requirements and their relationships. Each version would correctly represent the business data requirements and rules, but one version might be more concise, or "elegant", than the other. It is somewhat analogous to creating computer programming code to implement some process or function. One programmer might require 15 or 20 lines of code, where another more experienced programmer might achieve the same outcome in a more efficient, or "elegant", solution with 6 or 8 lines of code[58].

D.2.3 IDEF1X Data Modeling Standard.

There is more than one method of diagramming the structure and relationships of business data. However, the one that we will describe is based on the Entity-Relationship (ER) model that was first proposed in 1976 by Dr. Peter Chen as a method for graphically (visually) representing the structure of data in a relational database. And while it is not the only modeling method for accomplishing this end, it is perhaps the most widely employed, because it maps well to the relational model in that the ER constructs it represents are easy to understand and transform into relational database tables, relationships, and business rules[59]. There are several symbolic notational schemes such as Bachman, "crow's foot," IDEF1X, and others for representing data structures in a graphic Entity-Relationship Diagram (ERD). However, the scheme preferred and mandated by DoD in the DoD 8320 series of policies, regulations and procedure manuals is the IDEF1X version of ER modeling based on the Federal Information Processing Standard (FIPS) 184[60]. First we will define and examine the basic components and constructs of an IDEF1X ERD, and then we will see how they are represented in an ERD using the symbolic notation based on the FIPS 184 IDEF1X notation standard.

D.2.3.1 IDEF1X Entity-Relationship Diagram Components

In this section we will define and discuss the important features of a data model diagram which consists of three major components and a number of other aspects or concepts that relate to the three major components. The three major components are:

- **Entities**. Representations of real or abstract things (people, objects, places, events, ideas, combinations of things, etc.) that are recognized as the same type because they share the same characteristics, that can participate in the same relationships, and about which data is collected and stored. Entities can be classified as either independent or dependent (in some textbooks referred to as "strong" or "weak").

---

58 The variability in data modeling comes primarily from the difficulty in determining the level of abstraction needed when defining the entities about which data is to be collected. This in turn may be driven by the nature of the business. Thus, if we are designing a database that collects information on some sport teams, and where no requirement exists to actually identify the members of each team, then TEAM may be a good entity in our logical data model. However, if more detailed information about the team members in needed, then the entity TEAM may not be sufficient, and perhaps another one, more fundamental, such as PERSON may be required.

59 In fact, commercial tools such as ERwin permit the direct generation of relational schemas out of the IDEF1X specification corresponding to the logical data model.

60 In addition, see *Designing Quality Databases with IDEF1X Information Models, Thomas A. Bruce, Dorset House Publishing, New York, NY, 1992, for a comprehensive treatment of the IDEF1X data modeling methodology.*

- **Attributes**. Properties or characteristics that are common to some or all of the instances (or occurrences) of an entity. Attributes can be either identifying or descriptive of the entity.

- **Relationships**. Relationships are associations between two entities (or sometimes between instances of the same entity). The entity from which the association originates is the parent entity and the entity that terminates the association is the child entity. The type of association is governed by the rules of the activity or process that creates the requirement for the association, such as whether the relationship is mandatory or optional, identifying or non-identifying.

The definitions of the above data model components should be familiar to us as they were previously defined in section one of this appendix as they related to the design of a physical database. (Recall that a data element in a database table column is represented as an attribute in a logical data model). But here we also have added some additional information to our definitions such as "independent and dependent" entities, "identifying and descriptive" attributes, and relationships that can be "mandatory or optional" or "identifying or non-identifying". The meaning of these additional pieces of information will be made clear in the next section describing IDEF1X notational symbology. But before we get there, let's first define some of these other concepts and terms associated with the three major components that are necessary to understand in order to "read" and interpret an Entity-Relationship diagram. We will begin with relationships.

- **Classifying Relationships**. Relationships can be defined, or classified, by their degree, connectivity, cardinality, direction, existence, and type. Let's consider each of these in turn.

  - Degree. The degree of a relationship refers to the number of entities participating in the relationship. The FIPS 184 IDEF1X standard limits relationships to 2 degrees -- meaning between two entities -- a parent entity and a child entity. Some modeling methods do permit relationships of greater than 2 degrees. However, these are much more difficult to implement directly in a physical database and are usually decomposed into a series of binary relationships (2 degrees) for implementation.

  - Connectivity. Connectivity refers to how instances (rows, or records) of the parent entity and instances of the child entity of a relationship are mapped to each other. There are three basic possibilities: one-to-one, one-to-many, or many-to-many. One-to-one means that, at most, one instance of the parent entity can be associated with only one instance of the child entity. An example would be an entity, ORDER, to record customer orders and an entity, ORDER LINE DETAIL, to record the details of an order where the business has a rule that an order may only refer to one item or product. The relationship would be one-to-one as in, "An instance of ORDER may relate to only one instance of ORDER LINE DETAIL and an instance of ORDER LINE DETAIL may relate to only one instance of ORDER". One-to-many means that an instance of the parent entity can be associated with zero, one, or many instances of the child entity, but an instance of the child entity can only be associated with one instance of the parent entity. An example of this would be: "A MALE PERSON may be the father of zero, one or more children, but a CHILD can only have one MALE PERSON for a father". And finally, many-to-many means that an instance of the parent entity can be related to zero, one, or many instances of the child entity and vice versa. An example would be: "A TEACHER may have zero, one, or more students and a STUDENT may have zero, one, or more teachers".

- Cardinality. Cardinality refers to the actual number of instances that may participate in a relationship between two entities. An example would be a school that has a database with entities to record data about teachers, students, courses, classes, etc. with "business rules" that say: "A Class cannot be larger than 20 Students". "A Teacher may not teach more than three Courses per semester". "A Student may not enroll in more than five Courses in a semester". Then the cardinality of the relationship between a CLASS parent entity and a STUDENT child entity is 20. And between TEACHER and COURSE is 3, and so on.

- Direction. Direction refers to which end of a relationship is the parent and which is the child. It can depend on the type of entities in the relationship and the type of connectivity. In a one-to-one relationship between an independent entity and a dependent entity, the direction is always from the independent entity (the parent) to the dependent entity (the child). If both entities are independent, then the direction is arbitrary -- it doesn't make any difference. In one-to-many relationships the entity occurring once (the "1" end) is always the parent. And, in many-to-many relationships the direction again is arbitrary -- it doesn't make any difference.

- Existence. Existence refers to whether the existence of one instance in an entity in a relationship depends on the existence of an instance in the other entity in the relationship. The existence of an instance in an entity can be either optional or mandatory. An example of mandatory existence in the relationship between the CLASS entity and the STUDENT entity for the school example would be: "A CLASS must have at least one STUDENT". Thus, an instance of a CLASS cannot exist unless it is associated with at least one instance of a STUDENT that is assigned to the CLASS. An example of optional existence is the statement: "A TEACHER may be assigned to teach a COURSE". Instances of TEACHER can exist whether or not they have been assigned to a COURSE and vice versa. The clue to existence often lies in the verb (or verb phrase) that describes the business rule represented by the relationship, i.e., "must" is mandatory and "may" is optional.

- **Type.** Type refers to whether a relationship is identifying or non-identifying. An identifying relationship is one where the primary key of the parent entity that is shared with the child entity becomes part of the primary key of the child entity. A non-identifying relationship is where the shared primary key of the parent entity is not part of the primary key of the child entity.

- Entity Types. There are two fundamental types of entities, independent and dependent (or strong and weak in some textbooks).

    - Independent Entity. An independent (strong) entity is one that does not depend on any other entity to be able to identify instances of the entity (the rows or records of the table in a database). This means that the primary key attribute(s) of the entity are native attributes of the entity.

    - Dependent Entity. A dependent (weak) entity is one that does depend on another entity to be able to identify the instances of the dependent entity. This means that at least one of the primary key attributes of the dependent entity originated in another entity and is a foreign key contributed to the dependent entity through a relationship.

- Attribute Types. There are also two fundamental types of attributes, descriptive and identifying. Identifying attributes are also known as candidate keys, primary keys and alternate keys.

    - Descriptive Attribute. A descriptive attribute defines some fact, characteristic, or property, about an entity that may be common to all instances of the entity, and whose individual values may be descriptive of multiple instances of the entity. As an example, for the entity, CAR, the attribute, Car Color, may be one of the characteristics about instances of cars that we wish to record and store. And the value, "fire engine red", that is one of the valid values defined for the domain of Car Color, could clearly be descriptive of more than one CAR.

    - Identifying Attribute**.** An identifying attribute is an attribute, or combination of attributes, whose value in the case of a single attribute, or collective values in the case of multiple attributes, constitute unique identifiers of instances of an entity, or the columns of a database table. Identifying attributes of an entity are known as "candidate keys". A formal definition for a key is: ***a set of one or more attributes of an entity (or columns of a given database table) whose combined values are unique for all instances of the entity (or rows of the database table.)*** In professional data modeling literature, keys are classified and named, or labeled, in a number of different ways generally relating to the properties of a key. So, for example, in addition to candidate keys, keys can be labeled as primary keys, alternate keys, natural keys, business keys, atomic keys, composite keys, surrogate keys, synthetic keys, intelligent keys, pseudo keys, serial keys, technical keys, and so on. However, it is the **Primary Keys** of the entities in a logical data model, which are of critical importance in developing a specification that will guarantee the consistency and integrity of data in a database. *Remember-- the only data elements in a relational database that are ever shared between tables in the database are the data elements that constitute the primary keys of the tables*. So let's take a closer look at the desired properties of a primary key and at the distinctions between some of the descriptive names for keys described above.

- **<u>Properties of Primary Keys</u>**. Following are the desired properties of a primary key. The first three in boldfaced text are mandatory properties for which the absence of any one would result in compromising the consistency and integrity of data items in the database. The second three are desirable, but not necessarily essential properties.

  - **<u>Unique</u>:** all primary key values are unique; no duplicate primary key values are permitted to exist. (This is the fundamental basis of relational database theory that guarantees the ability of the database to uniquely identify and access the same row in a database table each time the primary key of the table is used in an instruction to the DBMS.)

  - **<u>Stable</u>:** the value(s) of primary keys, once created, are not permitted to change over time. (If key values were permitted to change, this would create the possibility of duplicate, or multiple, rows representing the same instance of an entity with different primary keys to exist in a database table.)

  - **<u>Mandatory</u>:** every entity instance <u>must</u> have a primary key value; the value(s) of primary keys are not permitted to be null. (This supports the relational model requirement that says no instance of an entity may exist unless it can be uniquely identified.)

  - <u>Unintelligent</u>**:** a primary key should not represent a business fact(s) beyond its role as a unique identifier of an entity instance, i.e., no embedded meaning.

  - <u>Minimal</u>: a primary key should be composed of the fewest possible number of attributes consistent with the requirement for uniqueness; single attribute keys are preferred.

  - <u>Available</u>: the means for creating unique primary keys must be readily available and easily accessible to support the creation of new entity instances as business needs require.

Now, what are the distinctions between the various labels that are given to database keys? The labels: primary, candidate, and alternate all relate to the state of consideration that is given to keys of an entity (table) during the process of selecting a primary key for the entity or table. This is because it is possible for an entity to have more than one attribute, or set of attributes, that might qualify as a key to identify unique instances of the entity, or the rows of a database table. An example of this might be a table for storing data items about materiel parts in a materiel management database. In such a table we might have columns for storing facts about instances of Vehicles such as the values of their National Stock Number, Vehicle Identification Number, Manufacturers' Serial Number, and Service Assigned Bumper Number, etc. Depending on the functional purpose of the database, any one of these attributes or some combination of these attributes might qualify as a unique identifier of individual instances of vehicles in the table.

During the initial stages of development, all attributes and groups of attributes that appear to have the necessary properties to qualify as unique entity instance identifiers are labeled candidate keys. As the development of the design matures, one specific attribute, or group of attributes, is selected by the designer to be designated as the **primary key** for the entity. (The selected primary key must satisfy all the mandatory properties of a primary key listed above.) Primary, candidate, and alternate keys may also be described or labeled with other names. As an example, a primary key can also be a natural key, business key, surrogate key, intelligent key, and so on.

These other labels for keys generally are descriptive of some characteristic of an attribute or column member of a key as related to one of the desired properties of a primary key. For example, if the value domain for an attribute represents specific business meaning other than its role as a row identifier, then it could be called a "business key". Examples of business keys in the world of DoD Logistics would be data elements such as National Stock Number, Manufacturer's Serial Number and the like. A business key is also sometimes labeled as a "natural" key because the values are "naturally" generated as a consequence of some functional business process, as in when serial numbers are stamped on manufactured parts. If the structure of a key is composed of more than one atomic data element, as in our example of National Stock Number in **Figure D.1.1-2 ,** then it would be labeled as an "intelligent" key because of the imbedded intelligence represented by the atomic data elements in the structure. Keys whose values do not represent any functional business meaning and whose sole purpose is only to uniquely identify an instance of an entity, or row of a database table, are labeled "**surrogate keys"**. Surrogate keys are also known as technical keys, pseudo keys, synthetic keys, and other similar synonyms for surrogate.

We have now arrived at the heart of the matter for enterprise keys because an enterprise key as described in this paper is a kind of **surrogate primary key.** Surrogate keys are usually represented as a meaningless number in a functional business sense. In addition, surrogate keys are defined to consist of a single attribute[61] which satisfies the minimal property for a primary key. And inasmuch as surrogate keys can be easily generated by the DBMS through a "next number" type function each time a new row is created and inserted into a database table, they also satisfy the last primary key property of availability.

When the DBMS is used to generate surrogate primary keys, it creates and maintains an internal table to keep track of the assignment of values of primary keys for each row of the tables in the database in order to prevent creating and assigning a primary key value that has already been used for a new row in a table. This table would consist of a column for recording the primary key values of every table in the database. If all primary keys in the database were structured as monotonically increasing numeric surrogate keys, then the table might look like Figure D.2.3-1.

---

[61] Bruce, Thomas A., *Designing Quality Databases with IDEF1X Information Models*, page 197, Dorset House Publishing, New York, NY, 1992.

PRIMARY KEY TABLE

Row No	Auto No	Driver No	Violation No
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
5		5	5
6			6

Figure D.2.3-1. Example of a DBMS Primary Key Assignment Table

By examining the primary key table in Figure D.2.3-1, we know that the database contains entries for five drivers, four autos, and at least 6 violations. Of course, for very large databases this Primary Key Table could become extremely large—possibly affecting the response time performance of the database to instructions and queries. The same function for assigning primary keys could also be accomplished by simply keeping track of the last value created for each primary key in the database. The table would then look like the graphic in Figure D.2.3-2.

PRIMARY KEY TABLE

Row No	Last Auto No	Last Driver No	Last Violation No
1	4	5	6

Figure D.2.3-2. Example of a "Last Value Assigned" Primary Key Table

Now if we wanted to create a row in the Auto table to add a record for another Auto, we simply have to access the Last Auto No data element in the PRIMARY KEY TABLE to find out that the last Primary Key value generated for the Auto table was "4", and we would just add 1 to that value to determine the Primary Key value of the new row to be "5". Then we would update the Last Auto No data item value from 4 to 5 in the PRIMARY KEY TABLE.

By now you should have picked up on the fact that the design and selection of primary keys are critically important components of a properly designed relational database and the logical data model that represents the specifications for the design. In the last section of this appendix we will examine some of the issues surrounding the implementation of enterprise surrogate primary keys into existing Army databases. But first we will present a brief tutorial on the symbolic notation employed in IDEF1X logical data models.

D.2.3.1.1 IDEF1X Logical Model ERD Symbolic Notation

IDEF1X is used for development and maintenance of both logical and physical data models. The IDEF1X notation for modeling an entity and its attributes in an ERD is a box representing a table of columns and rows of data values about the entity. Figure D.2.3-3 depicts the notation. The entity name is displayed above the box and the identifying and descriptive attributes are listed inside the box. A line divides the box into two sections. Identifying attributes, which comprise the Primary Key of the entity and the table it represents, are listed above the line. Descriptive, non-identifying attributes are listed below the line.

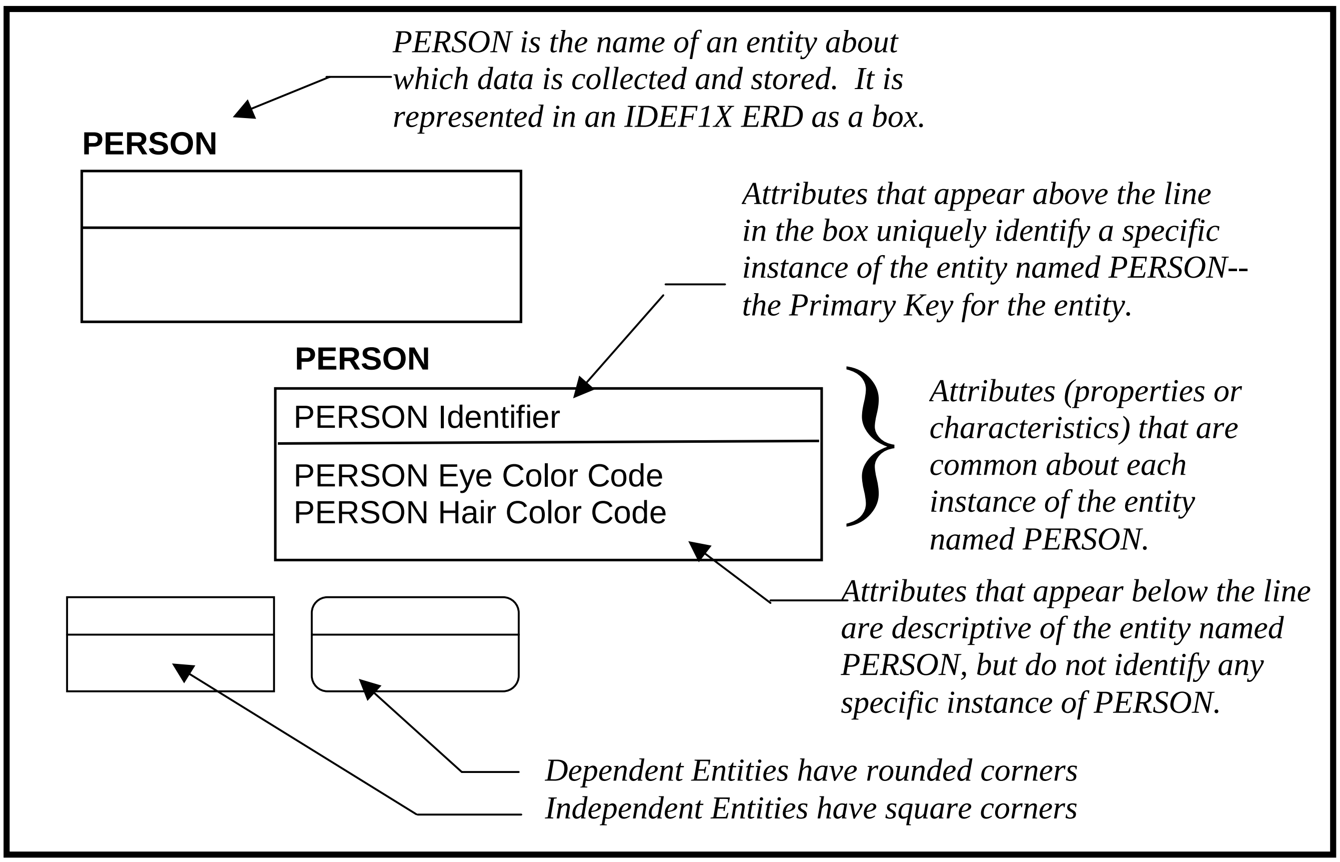
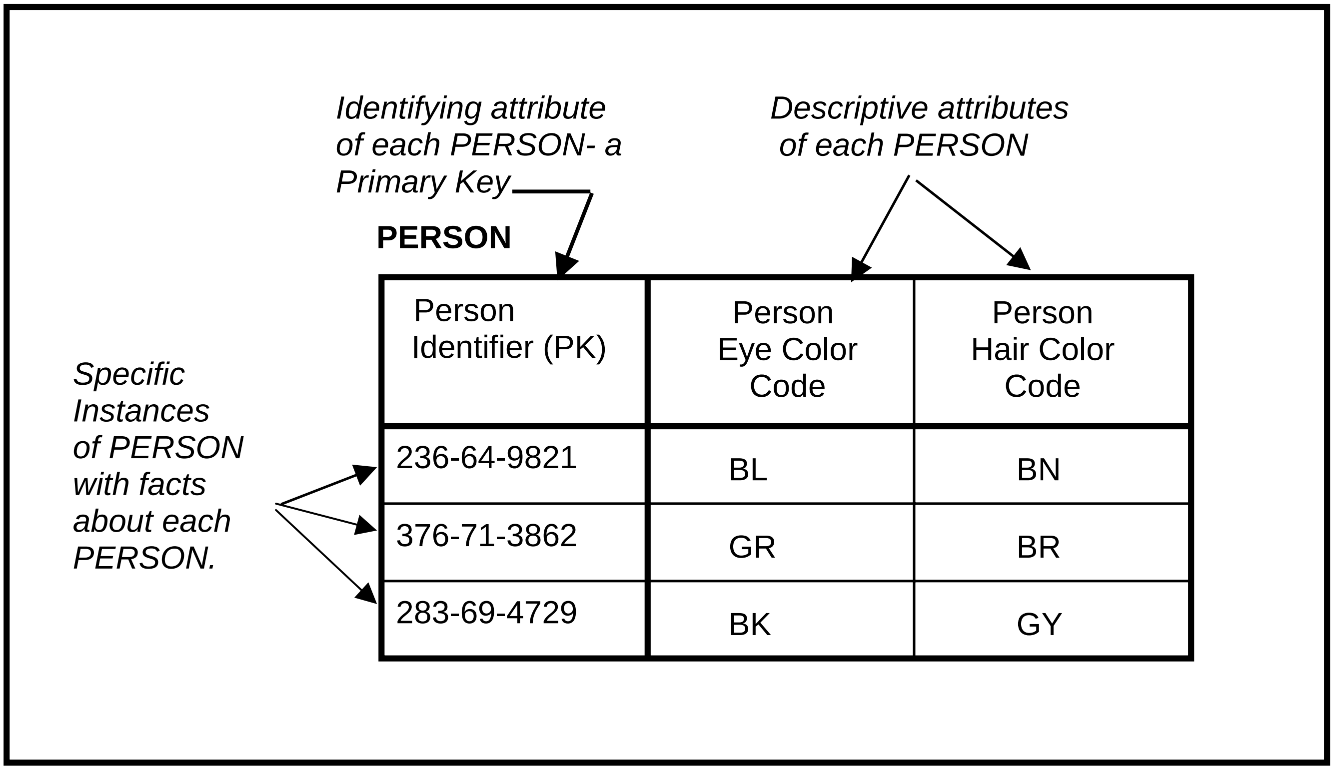


Figure D.2.3-3. IDEF1X ERD Entity – Attribute Representation

The entity, PERSON, in Figure D.2.3-3 specifies the structure of the table that might be required in a database to store data about persons. The entire table represents each occurrence of an entity such as PERSON. The rows of the table represent each unique instance of PERSON about which data is being collected and stored. Figure D.2.3-4 depicts the translation of a logical model entity, such as PERSON, into a physical database table.



Person Identifier (PK)	Person Eye Color Code	Person Hair Color Code
236-64-9821	BL	BN
376-71-3862	GR	BR
283-69-4729	BK	GY

Figure D.2.3-4. Example of an ERD Entity Represented as a Physical Database Table

Relationships between entities in a logical model are depicted by lines that connect one entity with another entity. Solid line connectors represent identifying relationships where the primary key attribute(s) of the parent entity that are shared with the child entity become part of the primary key attribute(s) of the child entity. Non-identifying relationships, where parent primary key attribute(s) are shared as below the line attribute(s) in the child entity, are represented by dashed connecting lines. Figure D.2.3-5 depicts the general notational constructs for representing relationships in an IDEF1X logical data model.

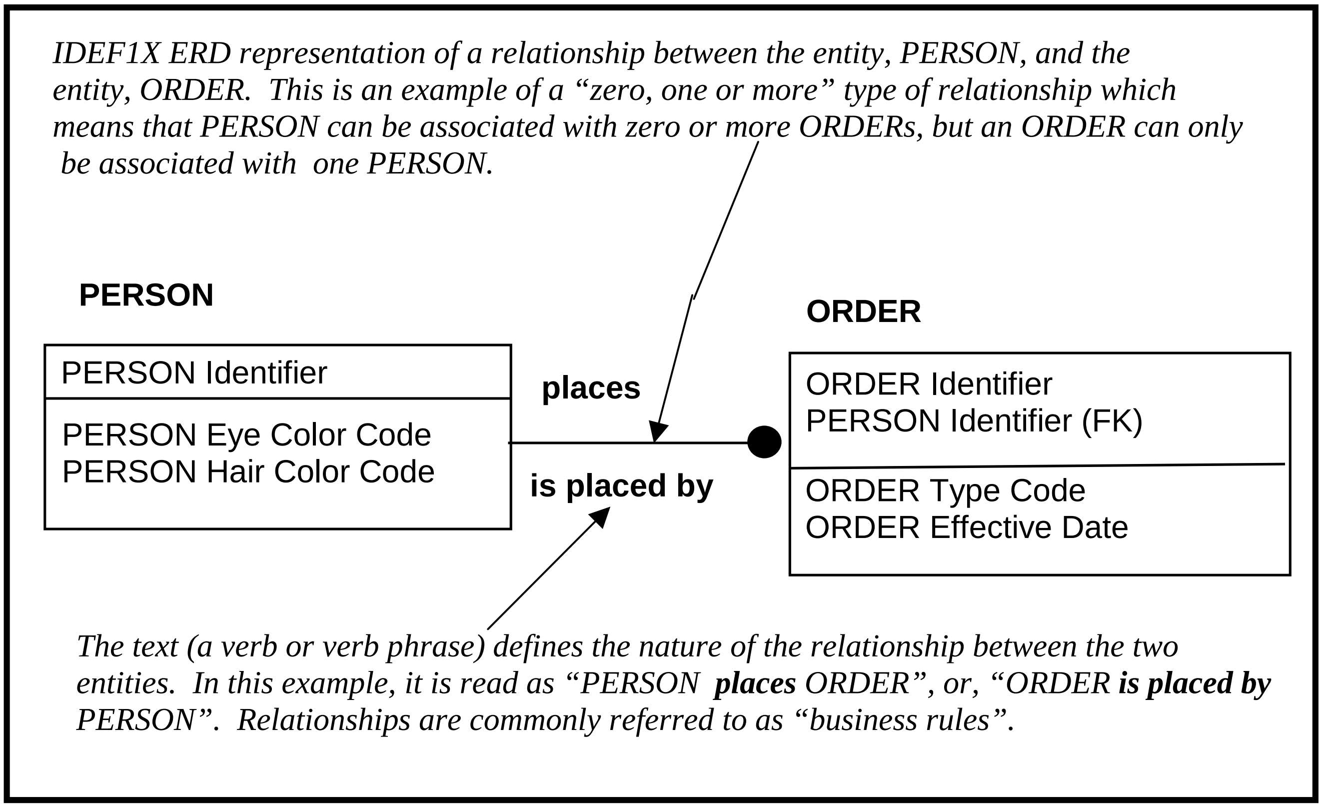


Figure D.2.3-5. IDEF1X ERD Representation of a Relationship

Notice in Figure D.2.3-5 that the primary key attribute of PERSON, the parent entity in the relationship, has become part of the primary key of ORDER through an identifying (solid line connector) relationship. The presence of the PERSON primary key, PERSON Identifier, in the ORDER entity is highlighted by the abbreviation (FK). FK is simply an abbreviation for "foreign key". This means that the attribute, PERSON Identifier, in the ORDER entity is not a native attribute of the ORDER entity, but is instead a ***"foreign key"*** attribute shared from the PERSON entity through their relationship. This reflects a functional business rule that states that every order must be associated with a customer that placed the order.

Figure D.2.3-6 depicts the IDEF1X symbolic notation for the types of relationships and their cardinalities that may be represented in a logical data model ERD. A database designer interprets these symbolic specifications and determines how to structure the database to comply with the "business rules" represented by the relationships. Let's look at a small portion of an actual logical model of a contracts administration activity in Figure D.2.3-7. Suppose our activity has a rule to assign identifying numbers to each warrant that was created during the course of business and to track the existence of those warrants in a database. But the organization also has a business rule that says that we may not create a warrant without also specifying the person who acts as contracting officer. These rules are expressed in the relationship notation between PERSON and CONTRACTING-OFFICER-WARRANT in Figure D.2.3-7.

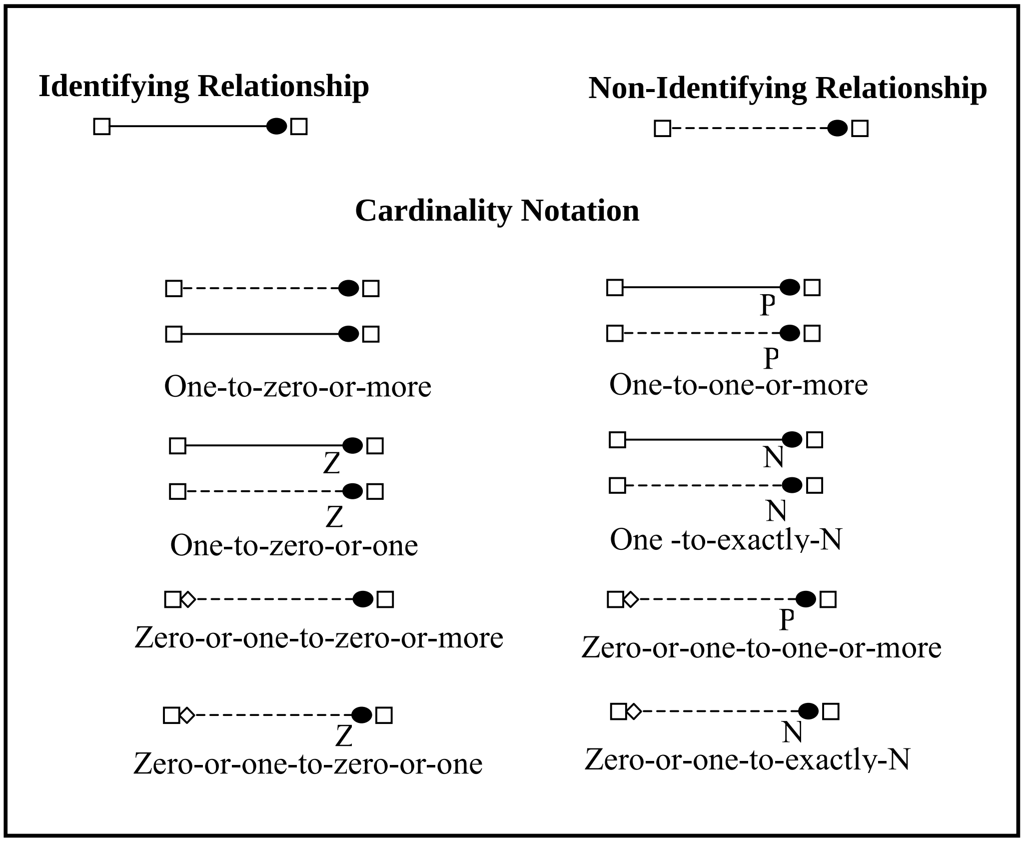


Figure D.2.3-6. IDEF1X Logical Data Model Relationship Notational Symbols

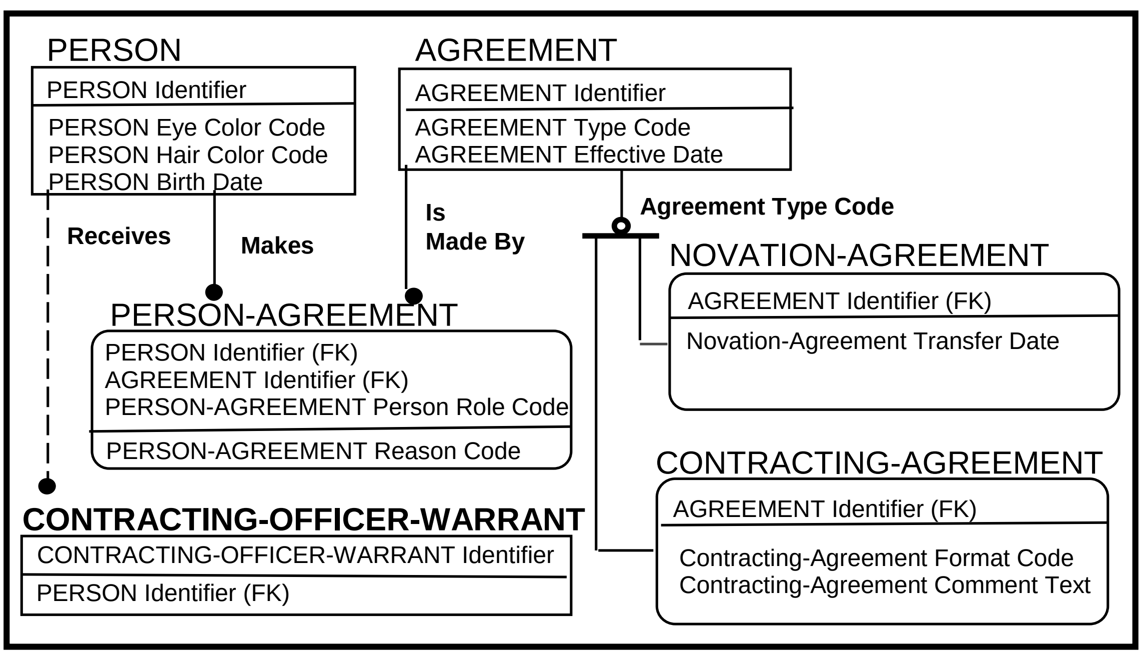


Figure D.2.3-7. Example of an IDEF1X Logical Data Model ERD

The dashed connecting line between PERSON and CONTRACTING-OFFICER-WARRANT denotes that PERSON shares its primary key, PERSON Identifier, with the CONTRACTING-OFFICER-WARRANT table, but the relationship is not identifying. So, PERSON Identifier appears as a foreign key below the line in CONTRACTING-OFFICER-WARRANTY TABLE. The absence of the diamond on the parent end of the relationship indicates that an instance of a CONTRACTING-OFFICER-WARRANT cannot exist unless it is also associated with a valid instance of a PERSON. If we attempt to enter a new warranty row/record (instance) into the CONTRACTING-OFFICER-WARRANTY table without also entering the value for a PERSON Identifier instance from the PERSON table, our DBMS DDL will prevent us from doing this. If, for some reason, we wished to delete a PERSON instance from the PERSON table in our database who still possessed a warrant (if the person retired), we would create the DDL code in our DBMS that would prevent us from doing that until the instance of the warrant associated with the person was first deleted from the CONTRACTING-OFFICER-WARRANT table. On the other hand, if we had a rule that permitted the creation of warrants to hold in a file until needed to be assigned to some contracting officer, then that rule would be indicated by adding the diamond to the parent end of the relationship. This would denote that the warrants listed in the CONTRACTING-OFFICER-WARRANT table exist independently and may or may not be assigned to a person.

These simple examples of how relationships are represented in a logical model are just the tip of the iceberg with respect to their importance in representing the business rules that will govern the degree of consistency and integrity of the data that is maintained in the database. Using the specifications contained in the logical data model, a database designer/architect can construct a specific physical database that is optimized for performance for a particular brand of DBMS such as Oracle, Microsoft SQL, Informix, Sybase, etc. The reader who desires to see the complete technical specifications of an IDEF1X logical data model should consult the *Federal Information Processing Standards Publication 184, Specification for Integration Definition for Information Modeling (IDEF1X).* However, for a thorough explanation of the methodology see the text by Thomas Bruce, *Designing Quality Databases with IDEF1X Information Models.*

D.2.3.2 IDEF1X Data Model Data Dictionary

The second part of a logical data model is the data dictionary that contains all of the data required to fully define and describe each entity and attribute in the ERD so that when the data is shared, both the sender and receiver of the data can be sure that the meaning of the transmitted data is the same for both of them. Recall that we have previously defined this kind of data as metadata, i.e., "data about data". Metadata includes items about a data element such as Data Name, Data Type, Data Definition, Data Steward, Functional Identifier, Maximum Character Length, Domain Definition, Domain Values, Authority Reference, High and Low Range Identifiers, etc.

Figure D.2.3.2-1 is an example of a Data Dictionary partial listing of the metadata that describes the attribute, or data element, Person Eye Color Code. In this example, the eye color is represented by a code such as BK or GR instead of the plain text Black or Gray. Now in this case, we might reasonably infer that BK means black, but does GR mean green or gray? The only way to be absolutely certain of the meaning conveyed by the values, BK and GR, we must consult the PERSON Eye Color Code item of metadata in the Data Dictionary named the Domain Value Identifier where we do indeed discover that BK represents "black" and GR represents "green" and not "gray". Gray is represented by the value, GY.

Element Name:	Person Eye Color Code
Access Name:	PR-EY-CLR-CD
Definition Text:	The code that represents the natural pigmentation of a person's iris.
Authority Reference:	U.S. Code title 10, chapter 55
Steward Name:	USD (P&R)
Domain Values:	BK. . . . . Black BL. . . . . Blue BR. . . . . Brown GR. . . . . Green GY. . . . . Gray HZ. . . . . Hazel VI. . . . . Violet

Figure D.2.3.2-1. Example of Data Model Data Dictionary Data Element Metadata

Metadata can include any item of information about a data element that is deemed necessary for information systems to be able to both share data, and to effectively and efficiently manage and maintain the data. Some metadata is mandatory and some is optional. In general, mandatory metadata items are those that are essential for effective data sharing and describe the meaning and structure of data elements such as data element definitions, data types, domain values, etc. Optional metadata tend to be those that are useful in managing the data such as functional area identifier, data steward name, authority reference text, security category, etc. Complete metadata requirements for DoD standardized data elements are listed and defined in DoD 8320.1-M-1 Data Standardization Procedures.

Effective management of metadata is essential for the development of data that is capable of being directly shared among information systems without the need for complex mediating data sharing software interfaces. It is the metadata about a data element that provides the required information that system developers or database designers need in order to be able to implement and use that data element in a shared information environment.

D.3 Implications for Enterprise Keys

So far, the entire discussion in this appendix concerning the structure and operation of relational databases and their specifications in logical data models has been from the point of view of a single database. We have seen how a relational database can record data items about an instance of an entity in a parent table and share access to those data items from any number of other child tables in the database by establishing relationships that are manifested by sharing the parent primary key with the child tables as foreign keys in the child tables. Any changes to the data items in an instance of the parent table are then immediately available to be accessed from any other table in the database that has a relationship with the parent table. The salient point is that the values of the data items are maintained only in the parent table—*entered once and shared many*.

We also heavily emphasized the importance of a primary key as a unique identifier of every instance or row of a database table. However, what we have so far described about the primary key of a table being a unique row identifier is true **only** for the table to which the primary key belongs. Let's look again at the PRIMARY KEY TABLE for our sample Department of Motor Vehicle database reproduced below in Figure D.3-1.

What we can see is that a data item value for a primary key in our database is unique only for a specific table. That is to say, for example, the value "1" appears as a data item for the Auto No primary key, and Driver No primary key, and Vehicle No primary key. The value "1" is unique in each primary key column in which it appears, but not across all primary key columns in the database. This condition is called "entity integrity,"[62] which means that the values of primary keys for tables in a database, as we have so far described them, are only designed to guarantee the integrity of the entity or table by enforcing the requirement that every primary key value for a row in a table be unique. Thus we ensure that duplicate, or redundant, rows cannot exist in a table. This is why we earlier said that the minimum requirements for identifying, or specifying, the "address" of any record (row) in a particular table in a database would be the table name and the primary key value. And similarly, to specify the address of a specific data item in a table requires the table name, column name, and primary key. Implicit in those specifications is the assumption that either the table names (identifiers) or the names (identifiers) of columns in the database, or both, are also unique within their domains. (If any part of an address is unique, then the entire address is unique). If we are dealing with a single database for which "we" are the designer/owner, this is probably a safe and fair assumption to make since we would have control of the designations of names (identifiers) over the structures in our database. Presumably we would ensure that all tables and columns in our database could be uniquely identified.

But suppose we could simplify the structure of the address for finding any row in any table in the database by just designating the identifier for a row without having to specify its table name and/or column name. This means that every row in every table in the database would require a primary key whose value would be unique across the entire database. This condition would be called "database integrity"[63]. One way to solve this problem of eliminating repeating values

[62] Riordan, Rebecca M., Designing Relational Database Systems, page 63, Microsoft Press, Redmond, WA, 1999.

[63] Also known as "object integrity". See the first article by Dr Tom Johnston cited in paragraph 2.2.3.1 of this paper.

across the primary keys of a database is to reserve blocks of values for the primary key of each table in the database. So in Figure D.3-1 the database designer might reserve 1 - 100,000 for Auto No, 100,001 - 200,000 for Driver No, and 200,001 - 300,000 for Violation No, and so on for other tables. However, what happens when you have underestimated the rate of growth for a table and it grows so large that you run out of numbers in the block? This problem has been encountered many times in the past. Solutions usually involve creating some sort of additional identifier that is concatenated to the original primary key values. However, these kinds of solutions are generally not very satisfactory because they only result in introducing unnecessary complexity to the effort required to manage updates and queries to the database, and greatly increase the risk of compromising the consistency and integrity of the data in the database.

PRIMARY KEY TABLE

Row No	Auto No	Driver No	Violation No
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
5		5	5
6			6

Figure D.3-1. Example of a DBMS Primary Key Assignment Table

Let's now consider an enterprise information environment composed of many information systems and many, many databases, each controlled by different organizations within the enterprise. This is the kind of information environment represented by the DoD "Enterprise" depicted in Figure D.3-2.

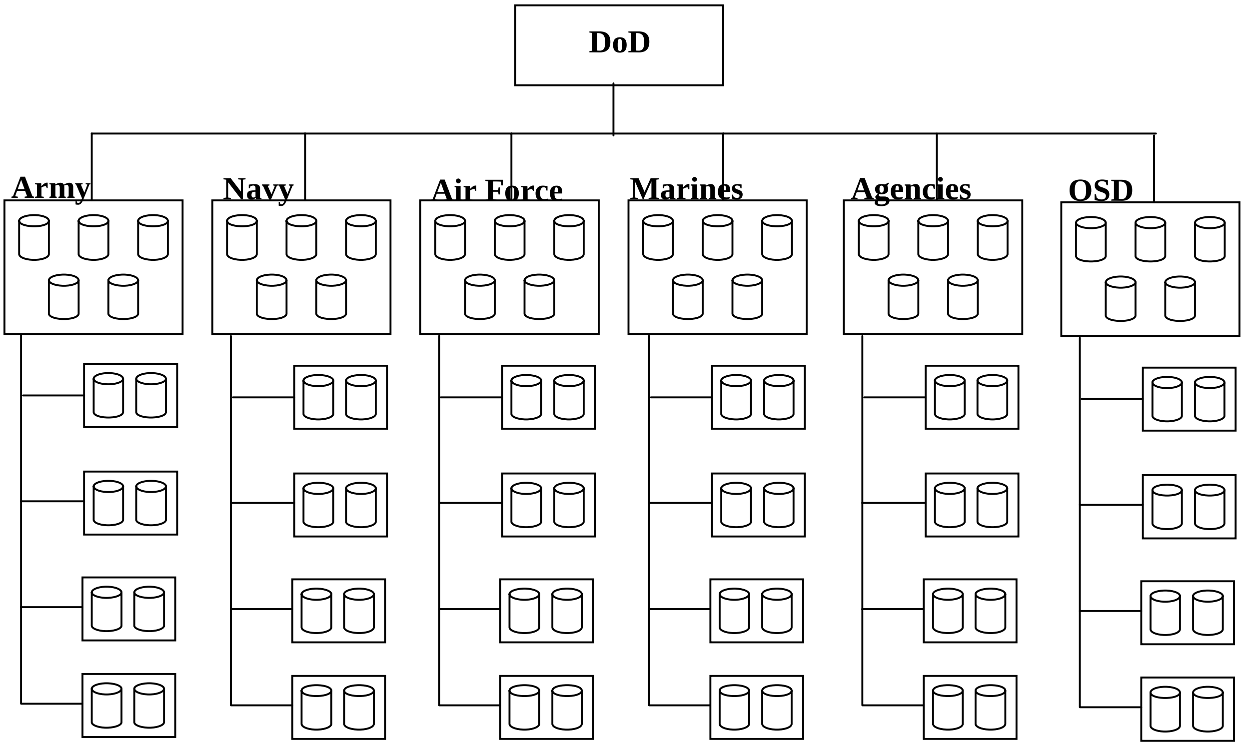


Figure D.3-2. Representation of the DoD Information Environment

There is no guarantee that in this kind of "enterprise" information environment that all table names and column names in all databases will be uniquely named and/or identified simply because there are so many, many different organizations responsible for designing and managing these databases. Therefore, the consistency and integrity of data items that are shared between databases in different information systems cannot be guaranteed without creating complex, expensive, maintenance intensive, mediating interface software between systems. But if we could extend the concept of "database integrity" one step further to the enterprise level by developing a scheme that would permit us to uniquely identify every row in every database table across multiple databases in an enterprise environment, we could create a state of "enterprise integrity" for the data relationships of the enterprise. What is really needed is a scheme that would permit every row in every table in every database across the enterprise to be uniquely identified with an enterprise primary key structure that would allow for enough unique values to be generated so as to almost guarantee never running out of values. If this could be accomplished, then the issue of requiring either table names and/or column names to also be unique across databases would be moot. That is exactly what an Enterprise Key as described in this paper will accomplish. ***The Enterprise Key concept not only guarantees that every row in a database can be uniquely identified, but will also guarantee that every row in every table in every database in every information system in the DoD information environment can be uniquely identified.*** This extends the concept of sharing access to data items from any table in a database through the sharing of primary keys beyond just a single database, but across all databases in an enterprise. This greatly reduces the risk of having redundant data items representing different values, which is another way of saying reducing the risk of inconsistent data item values existing within enterprise databases. Quality and accuracy of enterprise data would be greatly enhanced.

D.3.1 Implementation Issues

If a general technical approach is provided, then the issues that must be addressed in implementing the ORG-ID enterprise key concept as described in this paper are largely management related and not technical. From a management perspective, the principal issue that must be decided is what kind of management structure and policy framework will be adopted for controlling and issuing enterprise keys for every new row of data created in an enterprise database table. These issues were discussed in detail earlier in the Implementation Strategy section of this paper.

From a technical perspective, implementing an ORG-ID enterprise concept is not complex, involving only the creation of a server(s) and the communicating infrastructure that will generate unique enterprise primary key values in response to requests from authorized users. The method for accomplishing this is described in detail in the series of articles in *Data Management Review* by Dr. Tom Johnston cited earlier in paragraph 2.2.3.1 of this paper. However, there are three technically related questions about this concept that are usually raised that deserve some clarification. The first question is:

> *How can the value of every enterprise primary key be guaranteed to be unique? Won't we eventually run out of values similar to the situation described above when block values are reserved for primary keys of tables in a database?*

The theoretical answer to the question is—yes. There is a finite limit to the number of keys that can be generated by the key structure of two 32 bit concatenated values proposed in this paper. However, this limit is so large that for all practical purposes it can be ignored. The range of available values really depends on the number of servers that are created for issuing keys. This is a management issue. The lower bound of the range occurs when only a single Organization server is created to issue keys for the entire enterprise; then the number of available values is equal to 4.3×10^9 (4.3 billion). The upper bound occurs if we allow each of 4.3 billion organization servers to issue 4.3 billion keys which results in 18.49×10^{20} possible key values—a very, very large number. We should make clear that an Organization server refers to the software module that would generate and track the creation of enterprise keys. Given the computer technology available today, management could choose to host as many as all of the 4.3 billion possible Organization servers on a single hardware platform, or alternatively, have 4.3 billion platforms each hosting a single server. The point is that the range of options available to management in selecting the optimum strategy for managing enterprise keys is virtually limitless.

To put this issue into perspective: If across the entire DoD enterprise, new database records were being created as rows in tables at the rate of 100 rows per second, then at the lower bound of one organizational key server the key values would be exhausted in about 13.65 years. At the upper bound of 4.3 billion key servers the values would not be exhausted for at least 5.86 x 10^{12} years. And if the management policy envisioned as few as only ten organization key servers, it would still take at least 136 years to exhaust all values. So, this question is entirely within the control of management in deciding what kind of structure will be adopted for controlling and issuing enterprise keys.

The second question relates to the choice of a surrogate key as the structure of the enterprise key:

> *Some functional users might be concerned that implementing surrogate keys will make it much more difficult to construct queries to find data. This concern arises from the fact that surrogate keys do not contain any inherent business meaning, yet will replace the more meaningful "business keys," with which users are familiar.*

Figure D.3.1-1 illustrates this concern with a real-world Logistics database table .

MATERIEL-ITEM-SUPPLY

Materiel-Item-Supply National Stock Number Identifier (PK)	Materiel-Item-Supply End Item Indicator Code	Materiel-Item-Supply Critical Fit Indicator Code	Materiel-Item-Supply Price Amount	Additional Attributes →

Data Item Values

Figure D.3.1-1. A Table From a Logistics Database

The primary key attribute, National Stock Number of the MATERIEL-ITEM-SUPPLY table, is a common Logistics data element well known by logisticians everywhere in DoD and used by many functional processes in Logistics. This is an example of just one of many kinds of "business keys" that are used by DoD Logistics databases. If the current primary key attribute, National Stock Number, is replaced by a new enterprise primary key attribute, the old primary key attribute is simply relabeled as an alternate key. National Stock Number can still be used to construct queries of the database. So in that sense, nothing should change from the perspective of users who are accustomed to using National Stock Number in their database queries. The only difference will be that this former primary key will be replaced as the attribute that is migrated to child tables as the manifestation of the relationship that the MATERIEL-ITEM-SUPPLY table might have with other database tables. The revised MATERIEL-ITEM-SUPPLY table using an enterprise primary key is at Figure D.3.1-2.

MATERIEL-ITEM-SUPPLY

Materiel-Item-Supply Row Identifier (PK)	Materiel-Item-Supply National Stock Number Identifier (AK)	Materiel-Item-Supply End Item Indicator Code	Materiel-Item-Supply Critical Fit Indicator Code	Materiel-Item-Supply Price Amount	Additional Attributes
1					
2					
3					
4					
5					
6					
7					

Data Item Values

Figure D.3.1-2. Example of an Enterprise Key Integrated Into a Logistics Database Table

The third question is somewhat related to the second, but much broader reflecting how the nature of the structural organization of database objects (tables, primary keys, and relationships) in established (sometimes called "legacy") information systems can be affected by the introduction of surrogate enterprise keys as primary keys. The best way to demonstrate the issue is by looking at the structure of the logical data model of a portion of a current established data structure. Figure D.3.1-3 depicts such a structure that is an extract from the DoD Corporate Logistics Data Model representing data structures implemented in existing DoD operational databases. The pertinent issue is:

> *How the primary key attributes of the entities migrate from parent entities to child entities through each relationship and are represented as foreign key attributes in the primary keys of the child entities.*

Notice that in the structure in Figure D.3.1-3, this migration of foreign keys through the chain of identifying relationships, beginning with AIRCRAFT-TYPE, results in larger and larger composite primary keys of multiple attributes in each succeeding entity in the chain. There is nothing incorrect with this construct. It is a perfectly acceptable approach in the structure of a database that employs business, natural, or intelligent primary keys in the design of the database. However, when the strategy for representing the structure of all primary keys is changed to one of single attribute surrogate enterprise primary keys for all tables, this will drive a change in the nature of relationships between tables to only permitting non-identifying relationships. Figure D.3.1-4 demonstrates this change when we assume that the single attribute keys in each of the four entities are surrogate enterprise keys.

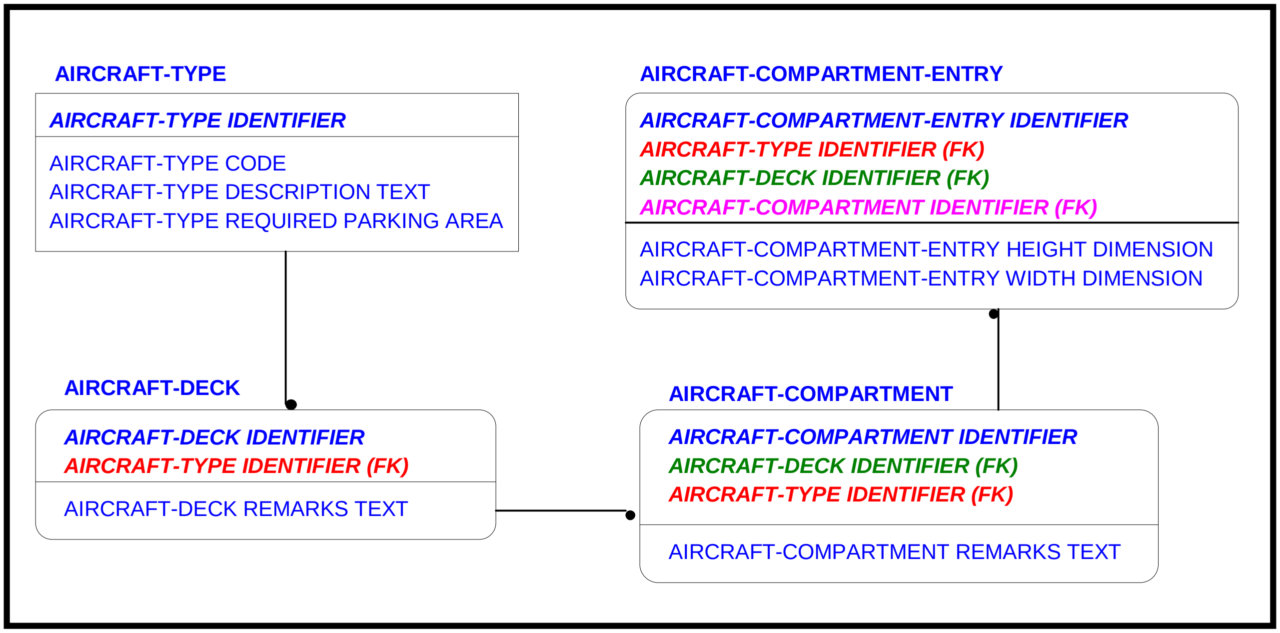


Figure D.3.1-3. Example Data Structure of a Legacy Database

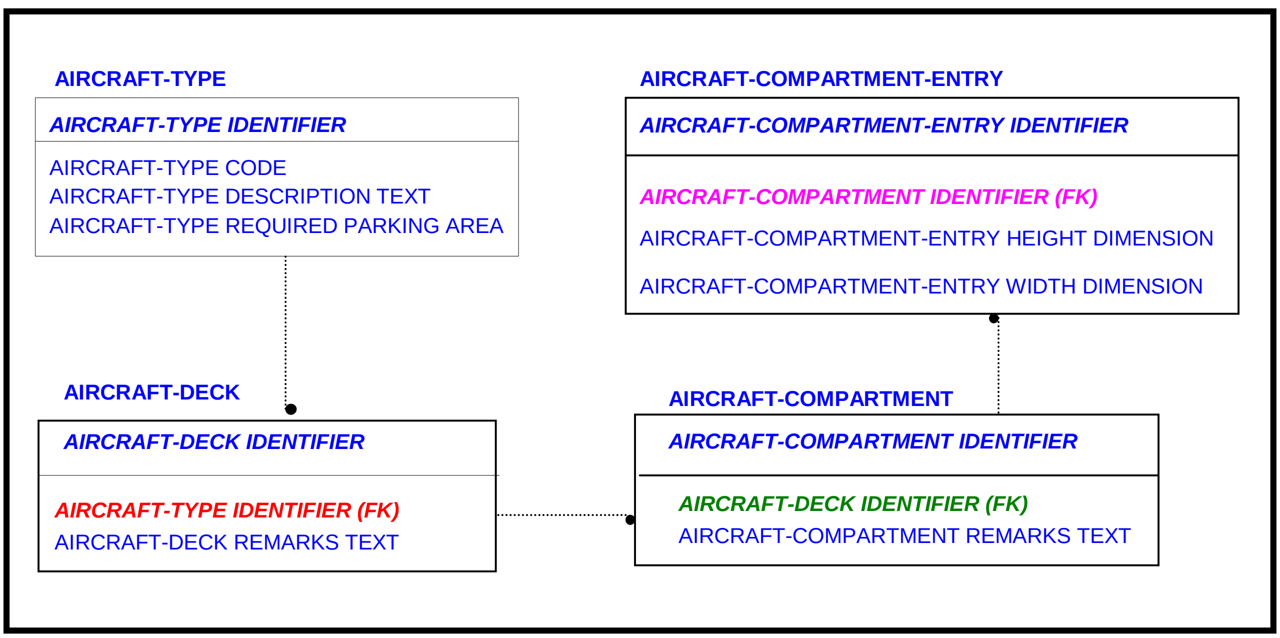


Figure D.3.1-4. Example Data Structure Based on Surrogate Enterprise Keys

The impact of this change in structure must be accounted for in the way business rules are enforced in the database engine to maintain the consistency and integrity of the data in the database. This will require some level of effort to analyze and implement changes in the DDL of the DBMS and to analyze the need for additional non-identifying relationships where it is deemed necessary to preserve the migration of attributes that were formerly implemented through identifying relationships. In addition, DML queries that depend on the presence of foreign key attributes in particular tables, but would be lost if enterprise keys were implemented, would have to be analyzed to determine the appropriate remedy, such as creating a non-identifying relationship to preserve the presence of the foreign key, or perhaps a restructuring of the query.

As noted above, implementing an enterprise key scheme into existing operational databases will require some analysis and work, but the task is not technically difficult to accomplish and can be accomplished in phases. Once the management issues have been decided, the first phase need only involve the creation of ORG-ID servers for generating enterprise keys and the insertion of additional columns into each table of a database to create the unique row identifiers. At this stage these enterprise key columns do not have to be implemented as primary keys. They can exist in their tables as just one more descriptive characteristic of the table in which they are resident. The conversion of the primary keys of the tables to enterprise keys can then take place at a deliberate pace in response to the progress of the impact analysis described above.

At this point we should note that while a proposal for using enterprise keys as universal identifiers of rows in tables among an integrated system of databases distributed across an enterprise is a "new" approach for improving integrity, accuracy, and interoperability of automated information systems, the concept, or the idea, of an enterprise key is not necessarily new. The syntax and functional purpose of an enterprise key as described in this paper identifies it as one kind of a Universal Resource Identifier (URI) which includes schemes for the unique identification of generic objects in an information space. The familiar Internet Uniform Resource Locator (URL) is another kind of URI, as is an Internet Protocol (IP) address of a node (machine) connected to the Internet[64].

Most leading professional database designers favor the use of surrogate primary keys over business keys for most situations. One of the disadvantages of using "business" keys as primary keys is that whenever the business process that the business key is related to changes, this often causes the values or the structures of the keys to change. This violates one of the mandatory properties of a primary key that says once created it should never change. The purpose of structuring enterprise primary keys as surrogate identifiers devoid of business meaning is to divorce enterprise keys from any pressure to change key values or structures whenever functional business processes may change. By avoiding the use of business keys, or as they are sometimes called, natural keys, the stability of the relationships between tables and the integrity and consistency of data items in the database is enhanced.

---

[64] For more information on URI's see RFC 1630, Universal Resource Identifiers in WWW, T. Berners-Lee, World Wide Web Consortium, found on the web at http//:www.landfield.com/rfcs/rfc1630.html.

APPENDIX E

DATA MODELING ISSUES

APPENDIX E – DATA MODELING ISSUES

E.1 Background

As mentioned in the body of this document, the primary objective of this study is to assess the applicability of using globally unique enterprise identifiers (EIDs) to manage both the materiel classes as well as the instances of such classes within the Army, and eventually DoD-wide, in a manner similar to what the Army has proposed for handling military units.

Let us assume that the introduction of centrally managed 64-bit integers to index all the materiel classes will become a reality, and that a scheme for handling the instances of these classes will be adopted (e.g., a concatenation of the globally unique ORGANIZATION-IDENTIFIER and another 32-bit integer sequentially generated by the EID Server of the organization in charge of that piece of materiel). Nevertheless, there remains the question as to whether the databases that the Army has been developing to support its information requirements contain the structures needed to allow the commander not only to build units on the fly, but to assign the necessary equipment to them, and keep track of the Logistics requirements associated therewith.

The next sections present a detailed assessment of the capability present in the data models listed below to generate Tables of Organization and Equipment (TOEs), and MTOEs, when using globally unique enterprise identifiers:

(1) the Land C2 Information Exchange Data Model (LC2IEDM)
(2) the Army Core Integrated Data Model (AICDM)
(3) the Army Joint Common Database (JCDB)
(4) the DoD Corporate Logistics Data Model (CLDM)
(5) the Logistics Integrated Database (LIDB), and
(6) the Army C4ISR Core Architecture Data Model (Army CADM).

E.2 Basic Concepts: Establishments and Holdings

The idea of specifying templates for the classes of materiel, and for keeping track of what is actually in hand, is not unique to the military. In the hotel industry, for example, it is customary to specify the kinds of furniture that rooms will contain, that is, the type of room furnishings, in accordance with the size and price of the rooms, and to keep track of what is actually put in the rooms via inventories. Ideally, the match should be one for one, but need not be necessarily so.

In the Army these templates, specifying not only the billets but also the type of personnel and the materiel classes required, are described in the Tables of Organization and Equipment (TOE). The DoD Data Architecture (DDA) adopted for its C2 portion the concepts that the Army developed to support its international land operations, namely ESTABLISHMENT for the templates, and HOLDING for the actual inventories. It should be noted that only classes can have cardinalities higher than one, whereas a real object exists only as one. In other words, there is only one instance of Joe Jones (an instance of PERSON with cardinality 'one'), but there can be 5 or more 'tank commanders' (an instance of PERSON-TYPE with cardinality 'five').

E.3 Representation of Authorized Lists of Resources and Actual Inventories in the LC2IEDM

In the LC2IEDM all battlefield objects can be specified as being either an instance of the five classes FACILITY-TYPE, FEATURE-TYPE, MATERIEL-TYPE, ORGANIZATION-TYPE and PERSON-TYPE, or as instances of a 'real' object, namely, FACILITY, FEATURE, MATERIEL, ORGANIZATION and PERSON. These classes and their realizations are further abstracted as being subtypes of either the entity OBJECT-TYPE or the entity OBJECT-ITEM. The relationship between these two super-types and their corresponding subtypes is shown in Figure E.3-1.

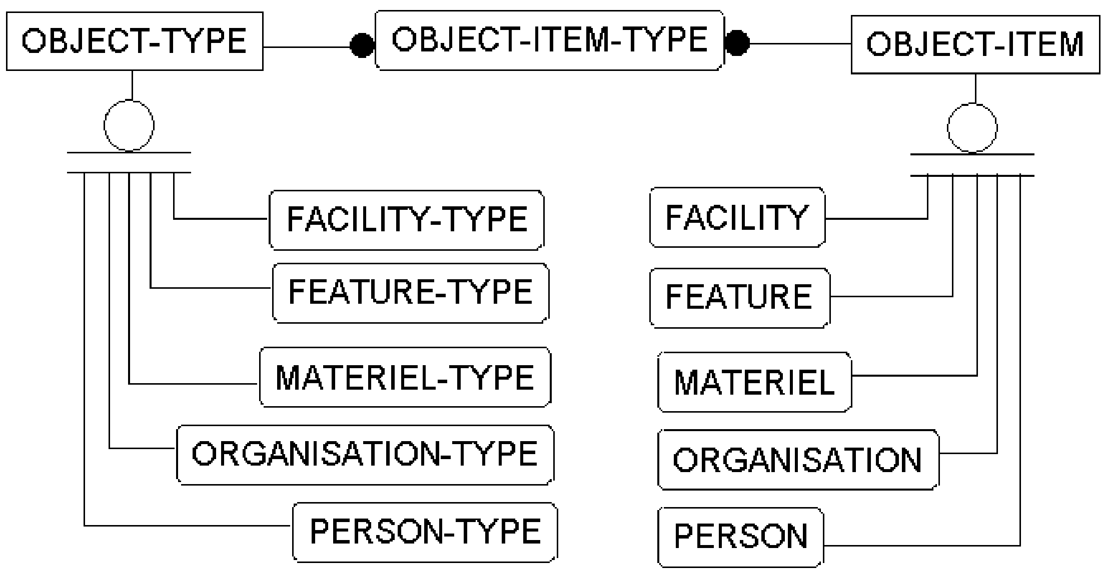


Figure E.3-1. Specification of the Battlefield Objects in LC2IEDM

E.3.1 Establishments Basic Concept

The conceptual structure of ESTABLISHMENT in the LC2IEDM is illustrated in Figure E.3-2. The cluster entity ESTABLISHMENT associates an OBJECT-TYPE with other OBJECT-TYPEs. The various components that make up that ESTABLISHMENT are represented in the cluster entity **ESTABLISHMENT-DETAIL**. An ESTABLISHMENT-DETAIL is the number of a specific OBJECT-TYPE authorized by a specific OBJECT-TYPE ESTABLISHMENT.

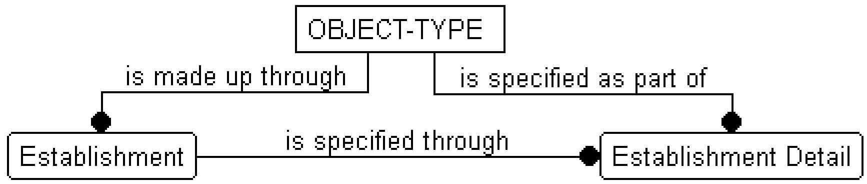


Figure E.3-2. The Concept of Establishment

The actual data structure is illustrated in Figure E.3-3, which shows all the ESTABLISHMENT entities in gray. There are two entities implementing the Establishment concept, both of which are supported by various Establishment Detail entities as follows:

- ORGANIZATION-TYPE-ESTABLISHMENT indicates what OBJECT-TYPEs an organization is supposed to have. These details are specified in several Establishment Detail entities as follows:

  (1) ORGANIZATION-TYPE-ESTABLISHMENT-**ORGANIZATION-TYPE-DETAIL** handling details pertaining to one ORGANIZATION-TYPE being composed of other ORGANIZATION-TYPEs.

  (2) ORGANIZATION-TYPE-ESTABLISHMENT-**PERSON-TYPE-DETAIL** that handles the data pertaining to types of persons belonging to an establishment (e.g. Sgt Infantry)

  (3) ORGANIZATION-TYPE-ESTABLISHMENT-**MATERIAL-TYPE-DETAIL** that handles the data pertaining to types of materiel belonging to an establishment (e.g. Tank Leopard 2)

- MATERIEL-TYPE-ESTABLISHMENT indicates what Materiel a specific MATERIEL-TYPE is supposed to consist of. A standard example would be an equipment parts hierarchy. The details would be stored in the **ESTABLISHMENT DETAIL** entity called MATERIEL-TYPE-ESTABLISHMENT-**MATERIEL-TYPE-DETAIL**.

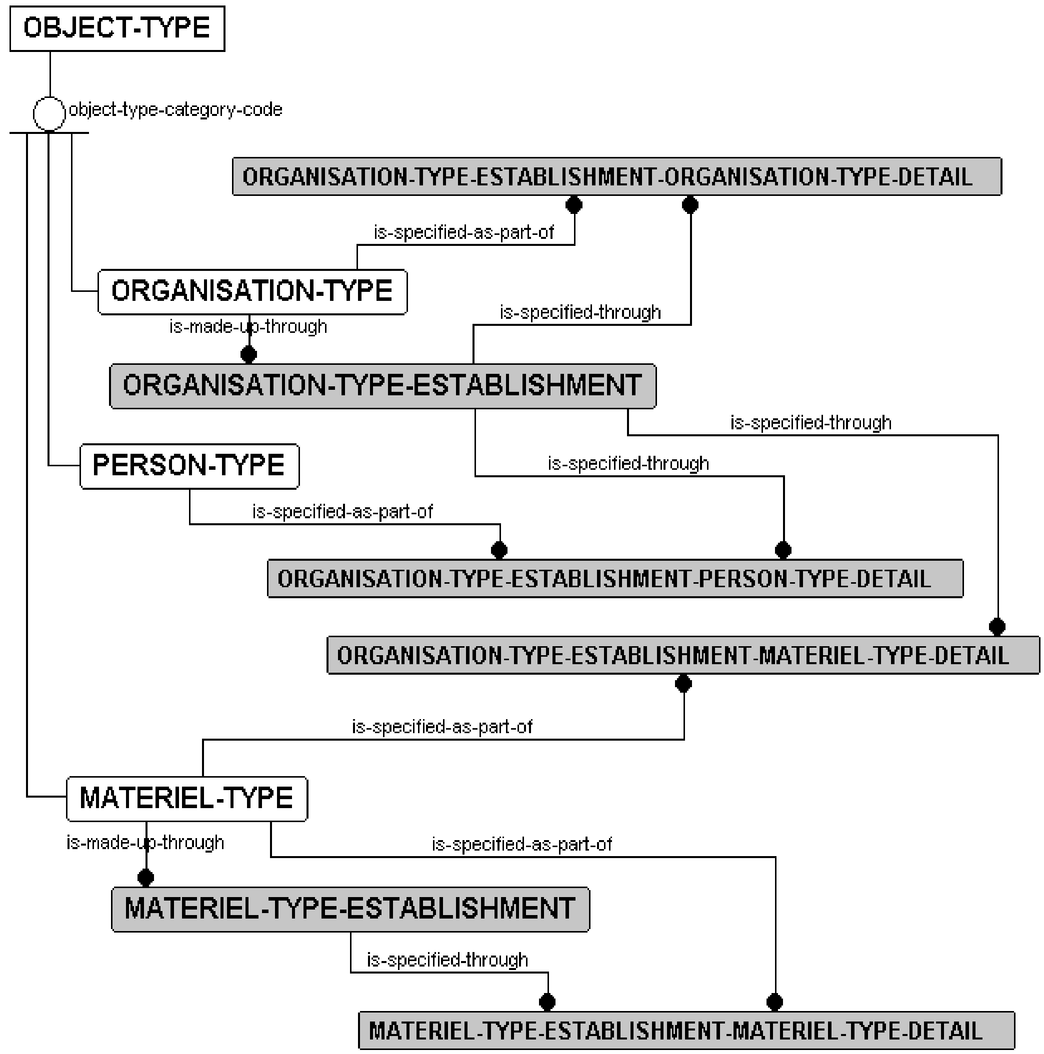


Figure E.3-3. Specifying Establishments for ORGANIZATION-TYPE and MATERIEL-TYPE

E.3.2 Assignment of Establishments to OBJECT-ITEMs

A particular OBJECT-TYPE may have more than one establishment at any given time, and a specific OBJECT-ITEM may have more than one establishment associated with it at a given time. This is handled by the use of MATERIEL-MATERIEL-TYPE-ESTABLISHMENT and ORGANIZATION-ORGANIZATION-TYPE-ESTABLISHMENT (see Figure E.3.1-1). This permits statements of the following kind: As of 1 March 1997, the 19th (US) Mechanized Division is assigned a specific Type Mechanized Division Establishment for war operations in a temperate climate. The establishment data structure (as described in the previous section) would detail the types and numbers of subordinate organizations, equipment, and personnel for that division.

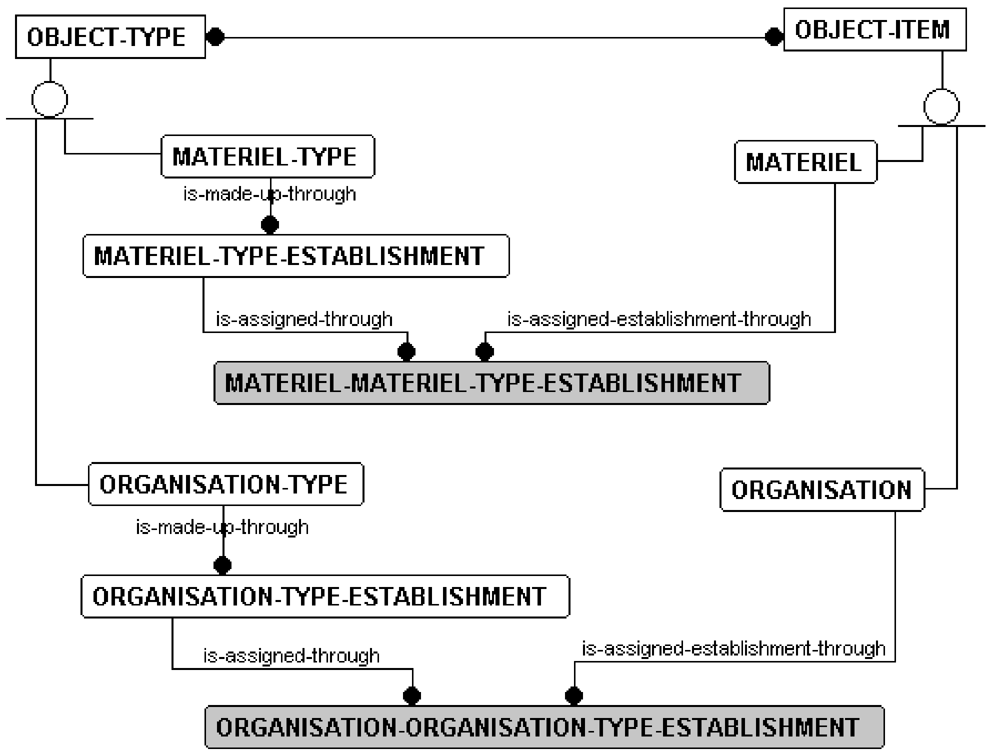


Figure E.3.1-1. Assigning Establishments to ORGANIZATION and MATERIEL

E.3.3 Holdings Basic Concept

OBJECT-ITEM and OBJECT-TYPE can be related through a HOLDING, showing how many of each OBJECT-TYPE are held by a specific OBJECT-ITEM together with the status of each. Thus, HOLDING allows quantities of an OBJECT-TYPE (e.g., Challenger MBT) to be attributed to a particular OBJECT-ITEM (e.g., a specific UK Armoured Regiment) that are at a particular status (e.g., operational) at a particular time. The structure of HOLDING is shown in Figure E.3.2-1.

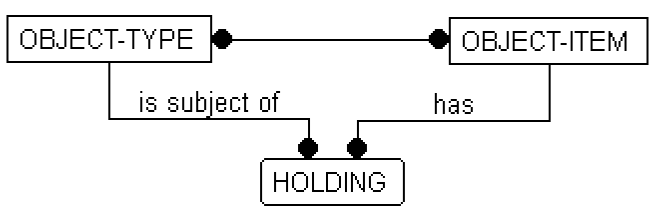


Figure E.3.2-1. The HOLDING Entity between OBJECT-ITEMs and OBJECT-TYPEs

Whereas an **ESTABLISHMENT** indicates what an organization or materiel is supposed to be composed of, the **HOLDING** concept captures what the organization or materiel actually contains. In other words, the difference between HOLDING and ESTABLISHMENT is that whereas ESTABLISHMENT details what an OBJECT-TYPE is authorized to have in terms of other OBJECT-

TYPEs, HOLDING details what an OBJECT-ITEM actually has (or is thought to have) at a particular time. For example, at a certain point in time, the 14th FR Engineer Regiment may have 300 regular troops, 100 drivers, 75 vehicles, 10 mine layers, and 16,000 mines. This concept enables the establishment of Logistics/personnel replenishment requirements as well as the assessment of organizational capability.

E.3.4 Holdings Detailed Structure

HOLDING is defined as the quantity of each specific OBJECT-TYPE that is held by, installed in, or included with a specific OBJECT-ITEM. The detailed structure for representing holdings in the specification is illustrated in Figure E.3.3-1. The entities OBJECT-TYPE, OBJECT-ITEM, and HOLDING correspond to those shown in the previous figure.

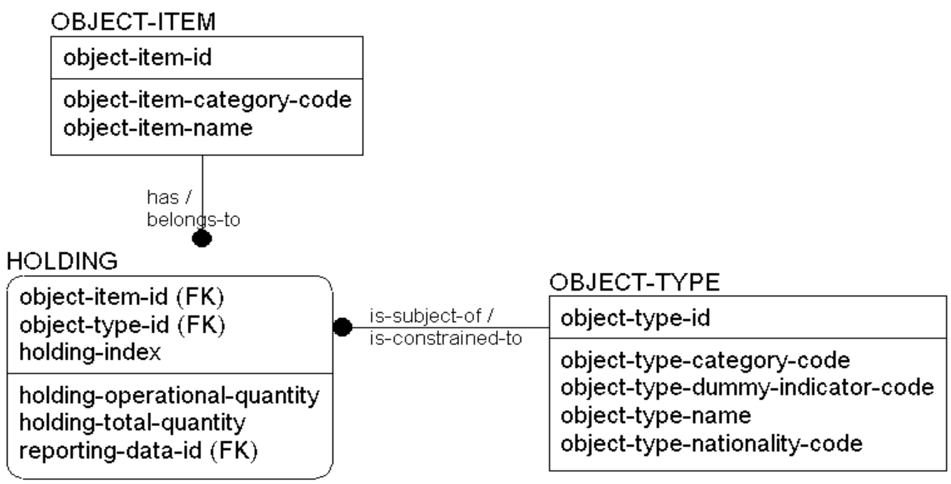


Figure E.3.3-1. HOLDING as a Relationship between OBJECT-TYPEs and OBJECT-ITEM

The attributes are:

(a) **object-item-id—**The unique value, or set of characters, assigned to represent a specific OBJECT-ITEM and to distinguish it from all other OBJECT-ITEMs.

(b) **object-type-id**—The unique value, or set of characters, assigned to represent a specific OBJECT-TYPE and to distinguish it from all other OBJECT-TYPEs.

(c) **holding-index**—The unique value, or set of characters, assigned to represent a specific HOLDING for a specific OBJECT-ITEM and a specific OBJECT-TYPE and to distinguish it from all other HOLDINGs for that OBJECT-ITEM and that OBJECT-TYPE.

(d) **holding-operational-quantity—**The non-monetary numeric value representing in a specific HOLDING the perceived count of specific OBJECT-TYPEs that are in operational status.

(e) **holding-total-quantity**—The non-monetary numeric value representing in a specific HOLDING the perceived total count of specific OBJECT-TYPEs.

(f) **reporting-data-id**—The unique value, or set of characters, assigned to represent a specific REPORTING-DATA and to distinguish it from all other REPORTING-DATAs.[65]

The purpose of the attributes is self-explanatory with the exception of holding-index. In order to cope with multiple estimates of holdings of the same OBJECT-ITEM, each holding has its own unique identifier (holding-index, a primary key attribute). The presence of this attribute permits a historical record of the holdings of an OBJECT-ITEM to be maintained.

The use of HOLDING is illustrated in **Table E.3.3-1**. The table lists the kind of records that may be maintained for one's own forces (in this case, two fictional units—a 101st (US) Mechanized Brigade and its parent, the 19th (US) Mechanized Division) and for reporting the estimated holdings of some opposing force units. The example for one's own force unit holdings lists several materiel items and the total military personnel assets.

Specifically in the first record, the holdings of OBJECT-ITEM 41822010 (which happens to be the 101st Mechanized Brigade) include the OBJECT-TYPE 214012 (which happens to be the M113 APC). The identifying information for OBJECT-ITEMs and OBJECT-TYPEs is included in this table only for purposes of illustration and ease of comprehension. In an actual database this information would be held in related tables. The record identifies the total holding quantity to be 44, of which 41 are operational. The last column contains a cross-reference to REPORTING-DATA, another data structure of the LC2IEDM, where the data about the holding estimate itself is specified. A previous version of this construct was called PERCEPTION, and it is now part of version 4.3 of JCDB. It should be noted that five records refer to a single reporting-data-id p1. The REPORTING-DATA data referenced through p1 would show that the reporting was done by the unit itself and that the holding is an actual count of both equipment and personnel. This example equates in broad terms to a normal unit situation report of major equipment and personnel holdings.

Three additional entries for the 101st Brigade have an index 41000002, since these are updates for several items reported earlier, namely, BFV, M1A1, and military personnel. All of these refer to reporting-data-id p20 that could correspond to the next reporting cycle.[67]

The holdings of the 19th Division follow the same table format. However, it should be noted that the initial Division holdings refer to two different reporting-data-ids: p5 and p6. Two instances

[65] Since each instance of HOLDING is an estimate, REPORTING-DATA provides the amplifying information about that estimate. Part of the amplifying information identifies the organization that is reporting the HOLDING, since the estimate is likely to have little value unless its source is known. The details of REPORTING-DATA entity are described in the full specification of the LC2IEDM, and do not form part of this report.

[66] Since each instance of HOLDING is an estimate, REPORTING-DATA provides the amplifying information about that estimate. Part of the amplifying information identifies the organization that is reporting the HOLDING, since the estimate is likely to have little value unless its source is known. The details of REPORTING-DATA entity are described in the full specification of the LC2IEDM, and do not form part of this report.

[67] The update on an OBJECT-TYPE in the example is placed immediately after the original entry for ease in reading and comparing. This could correspond to the way the operational user requires the display to be organized. The actual database table order is immaterial to the user, since the data is accessed via SQL queries through the key attributes, and displayed in appropriately formatted screens.

of REPORTING-DATA are required in this case since the reporting was done at one time but the effective times for the holdings are different. The divisional data also contains updates, one of which shows the decrease in the number of operational AH-64 attack helicopters from 26 to 24. The change in divisional personnel includes both a decrease in the number of operationally ready personnel and an increase in the total holdings of military personnel. The latter could be due to arrival of replacements.

Table E.3.3-1. Illustration of HOLDING

HOLDING

object-item-id	object-type-id	holding-index	holding-operational-quantity	holding-total-quantity	reporting-data-id
41822010 [101 Mech Bde]	214012 [Armoured Personnel Carrier, M113]	41000001	41	44	p1
41822010 [101 Mech Bde]	214015 [Bradley Fighting Vehicle (BFV), M-2]	41000001	94	102	p1
41822010 [101 Mech Bde]	214015 [Bradley Fighting Vehicle (BFV), M-2]	41000002	88	102	p20
41822010 [101 Mech Bde]	220012 [Howitzer SP, 155mm, M-109]	41000001	22	24	p1
41822010 [101 Mech Bde]	229005 [Tank, Main Battle, M1A1]	41000001	55	58	p1
41822010 [101 Mech Bde]	229005 [Tank, Main Battle, M1A1]	41000002	49	58	p20
41822010 [101 Mech Bde]	41600 [Military person-type]	41000001	3607	3943	p1
41822010 [101 Mech Bde]	41600 [Military person-type]	41000002	3511	3943	p20
41822012 [19 (US) Mech Div]	212010 [AD Missile System, HAWK]	41000001	6	6	p5
41822012 [19 (US) Mech Div]	221001 [Helicopter, Attack, AH-64 (APACHE)]	41000001	26	26	p5
41822012 [19 (US) Mech Div]	221001 [Helicopter, Attack, AH-64 (APACHE)]	41000002	24	26	p21
41822012 [19 (US) Mech Div]	229005 [Tank, Main Battle, M1A1]	41000001	280	290	p6
41822012 [19 (US) Mech Div]	229005 [Tank, Main Battle, M1A1]	41000002	265	290	p21
41822012 [19 (US) Mech Div]	323001 [Multiple Launch Rocket System (MLRS)]	41000001	9	9	p5
41822012 [19 (US) Mech Div]	41600 [Military person-type]	41000001	15227	15569	p5
41822012 [19 (US) Mech Div]	41600 [Military person-type]	41000002	15202	15711	p28

E.3.5 Business Rules

A fundamental challenge in maintaining organizational holding information for echeloned formations is the treatment of aggregation in moving from the lower echelons to the higher. To provide a uniform way of specifying holdings the following rule is postulated:

The total or operational quantity entered for an OBJECT-TYPE in a HOLDING of an ORGANIZATION at any echelon is taken to be the aggregation of all OBJECT-TYPEs held by that ORGANIZATION and all organizations at echelons below.

Thus, the holding of PERSON-TYPEs by a brigade will include all the PERSON-TYPEs not only in the brigade headquarters and headquarters company but also those in all its constituent units. Likewise, the holdings of tracked vehicles comprises all those in the brigade manoeuvre battalions as well as the supporting units and the brigade headquarters. It is anticipated that the staff at each echelon level will enter the holding quantities.

One of the advantages of the rule in the previous paragraph is that the reporting of holdings at any echelon does not depend on the availability of data from the lower echelon levels. It assures that an entry is made at each echelon regardless of the status of reporting below that echelon level, even if the staff must make estimates of holdings for units below its own echelon. Furthermore, even if the adopted database-to-database replication contract for holdings is filtered by omitting data for echelons below a certain level, the recipient will not be deprived of vital information.

The OBJECT-TYPE under which a holding is reported may change with echelon to reflect differing needs with respect to the granularity of data. The example in Figure E.3.4-1 illustrates the concept. Battalion ABC has three companies under its command each with its own complement of equipment. The battalion itself has 6 APCs in its table of equipment. Sub-table (a) shows the holdings for individual units. Each company reports to the battalion its holdings as shown in the first three records. In sub-table (b), the battalion may report its total holdings at the detailed level that is shown as Option 1. In this case, each type of equipment is reported separately. On the other hand, the battalion may report its total holdings aggregated at the tank and APC level. The latter option is shown as Option 2. The choice of the type of reporting is based on operational considerations or command guidance.

Sub-table (a) Unit Holdings

Unit	Equipment Type	Total Number	Number Operational
Coy A	Tank Type 1	16	15
Coy B	Tank Type 2	12	10
Coy C	APC Type 17	18	15
Bn ABC	APC Type 11	6	5

Sub-table (b) Holdings Reported by Battalion

Option 1

Equipment Type	Total Number	Number Operational
Tank Type 1	16	15
Tank Type 2	12	10
APC Type 17	18	15
APC Type 11	6	5

Option 2

Equipment Type	Total Number	Number Operational
Tank	28	25
APC	24	20

Figure E.3.4-1. Options for Reporting Holdings at Higher Echelon

E.4 Representation of Authorized Lists of Resources and Actual Inventories in the AICDM

Beginning in 1997, the Army undertook a complete review of the C2 portion of the DDA with the intent to ensure that all the proposed structures could be readily implemented by those Army agencies building new information systems. In addition, the Army also began to integrate other portions of its information infrastructure into the overall standardized data model for C2. This model, which takes as its point of departure the C2 Core Data Model (C2CDM), and expands it to accommodate the newly identified data requirements, was named the Army Integrated Core Data Model (AICDM).

E.4.1 Establishments

One of the major differences between the LC2IEDM and the AICDM is that the latter does not contain the high-level entities OBJECT-ITEM and OBJECT-TYPE. Instead, all ten battlefield objects are treated as independent entities. As a result, the ability to manage associations between classes through a few tables corresponding to the respective super-types is no longer possible. Instead, each one of the battlefield object classes gets treated separately and explicitly shows which associations it supports. The positive side of this approach is that all rules are made more explicit than in the LC2IEDM. On the other hand, the implementation requires many more tables and joins, and the key management is more complex than in the LC2IEDM.

E.4.1.1 ORGANIZATION-TYPE ESTABLISHMENTS

The current structures of the AICDM shown in Figure E.4.2.2-1 allow for a given class of ORGANIZATION to have associations with instances of FACILITY-TYPE, MATERIEL-TYPE, ORGANIZATION-TYPE, and PERSON-TYPE. Thus, the only class of battlefield objects that cannot be related to ORGANIZATION-TYPE is FEATURE-TYPE (e.g., rendezvous area types, air corridor types, geographic feature types).

For the purposes of this study, it is clear that the AICDM has the required structures to support not only the decomposition of classes of large military units into their constitutive classes of subunits, but it can also link classes of materiel to an instance of ORGANIZATION-TYPE, as well as types of personnel. This capability supports the three major components – organization, materiel, and personnel -- necessary to build units on the fly, as well as to specify default templates in accordance with current Army doctrine.

Additionally, the AICDM also permits the allocation of classes of FACILITY to a given class of ORGANIZATION, in case such templates may be required for planning operations and managing battlefield resources.

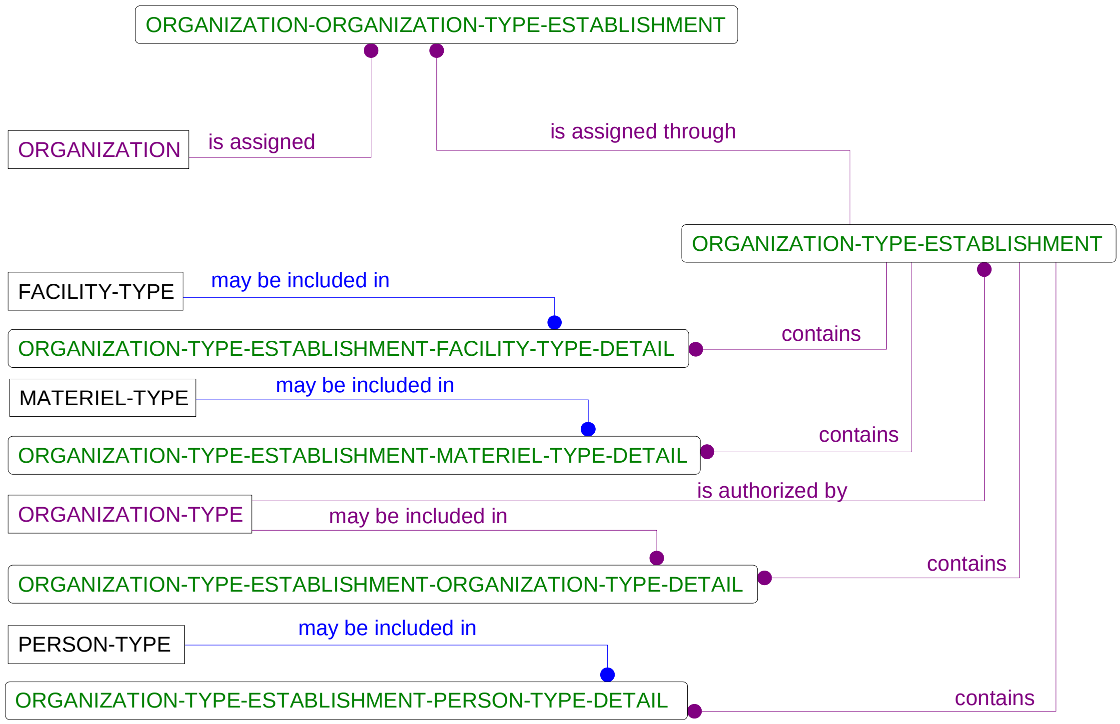


Figure E.4.1.1-1. AICDM Establishments for ORGANIZATION

E.4.1.2 PERSON-TYPE ESTABLISHMENTS

The AICDM also contains a data structure that allows the commander to specify the classes of MATERIEL that the relevant classes of personnel may be required to have. In this manner it is not only possible to document what a given class of billet requires, but also what is the class of equipment that the different types of soldiers must carry for a given type of climate or combat scenario.

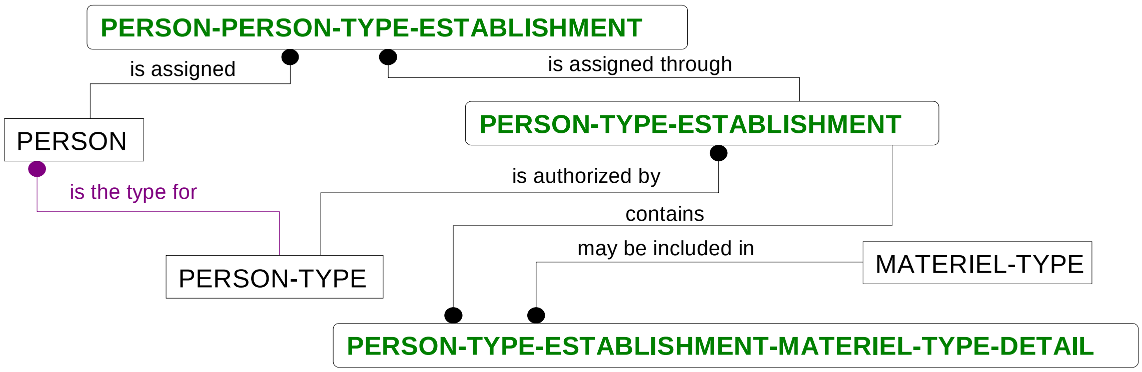


Figure E.4.1.2-2. AICDM Establishments for PERSON

E.4.1.3 MATERIEL-TYPE ESTABLISHMENTS

In the AICDM it is possible to specify a full decomposition of a class of materiel in terms of sub-classes of materiel. This way, logisticians could build complete descriptions of the parts that weapon systems consist of for the purpose of their maintenance and repair. In addition, it is also possible to link the classes of personnel that may be required to operate complex pieces of materiel.

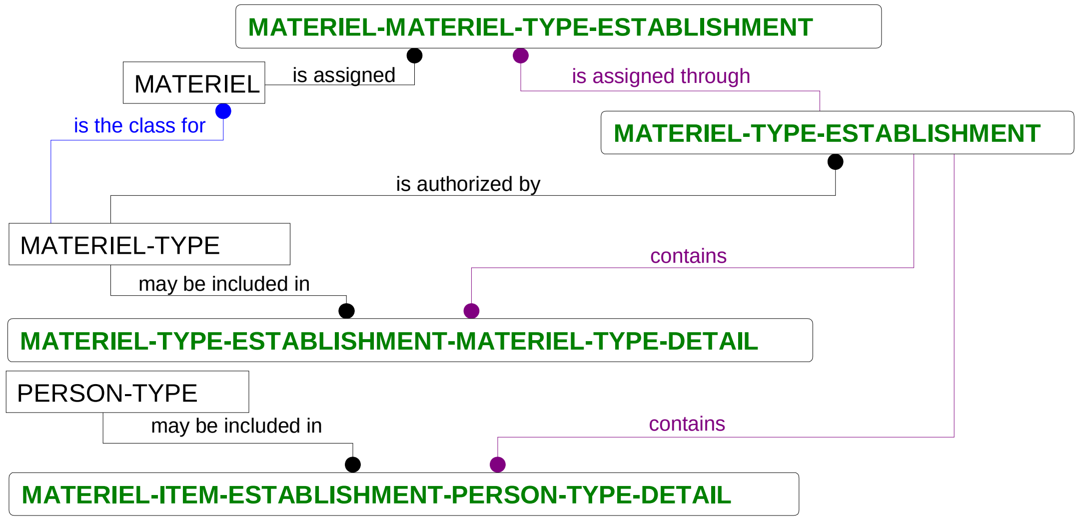


Figure E.4.1.3-1. AICDM Establishments for MATERIEL

E.4.1.4 FACILITY-TYPE ESTABLISHMENTS

Although the present study is built on the idea of an organization-centric view of the Army, this is not the only possible way to relate the classes of battlefield objects among themselves. In particular, the AICDM provides all the data structures necessary to relate classes of FACILITY, FEATURE, ORGANIZATION, MATERIEL, and PERSON to classes of FACILITY.

These structures may not be as pertinent to the way in which the commander needs to manage his resources during the prosecution of the battle, but they provide a robust specification for planners dealing with sustaining base activities, such as the planning and costing of the whole communication infrastructure for the Army bases throughout CONUS.

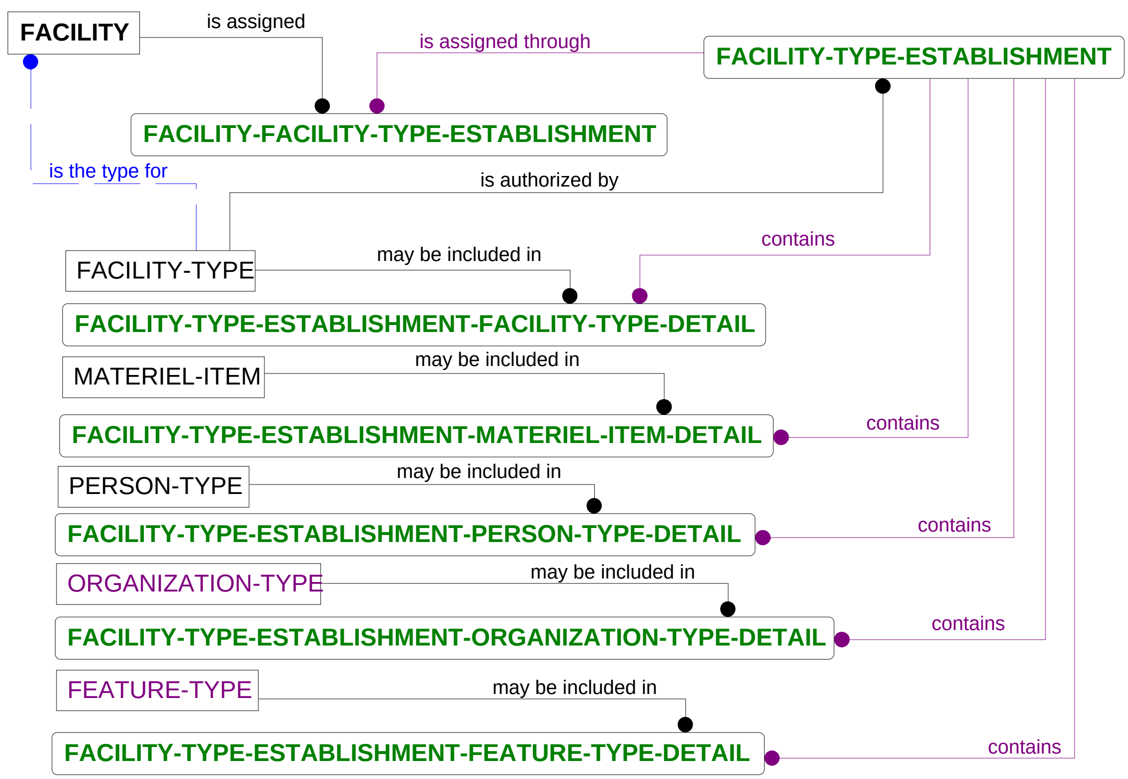


Figure E.4.1.4-1. AICDM Establishments for FACILITY

E.4.1.5 FEATURE-TYPE ESTABLISHMENTS

The last specification of how classes of battlefield objects relate to other classes of battlefield objects in the AICDM is the FEATURE-TYPE-ESTABLISHMENT, which permits the association of instances of FEATURE-TYPE both to other instances of FEATURE-TYPE, as well as FACILITY-TYPE. In this manner, it is possible to specify classes of complex control features such as the type of air space corridors needed for a class of air control management, and decompose them at the required level of detail.

In addition, the AICDM also allows the specification of the classes of FACILITY that may be required for a given class of FEATURE.

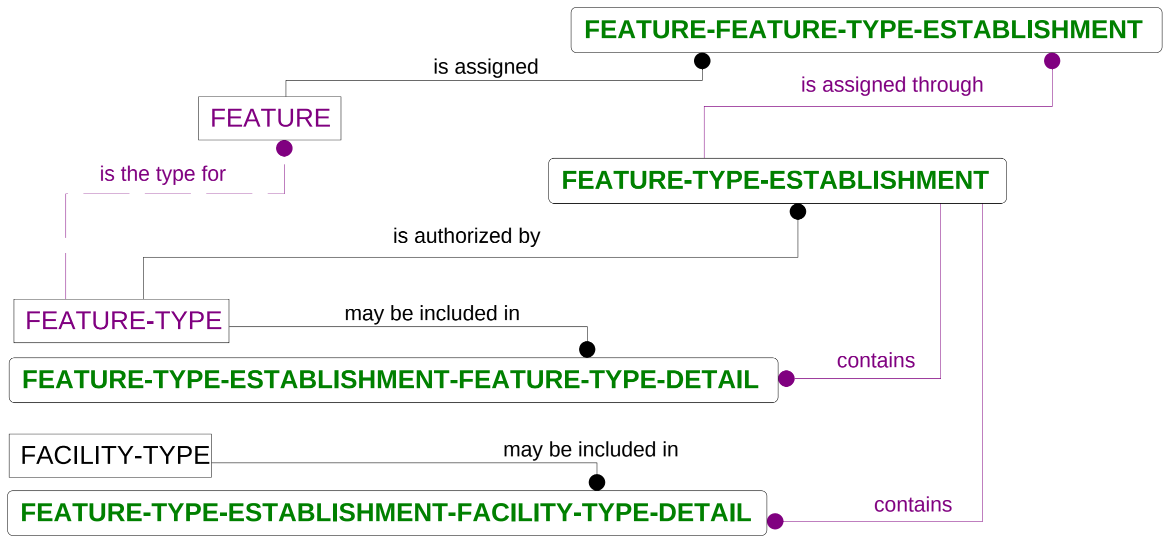


Figure E.4.1.5-1. AICDM Establishments for FEATURE

E.4.2 Holdings

The AICDM contains data structures, shown schematically in Figure E.4.2.1-1, that permit the specification of the inventories held by all five battlefield objects with respect to all or some of the other battlefield classes of objects. Specifically, in the current version of the AICDM the instances of FACILITY, FEATURE and ORGANIZATION are capable of having inventories of all five classes of battlefield objects. In contrast, instances of MATERIEL can only have inventories of materiel classes and personnel classes, and PERSON can only have inventories of materiel classes and nothing else.

E.4.2.1 Holdings Detailed Structure

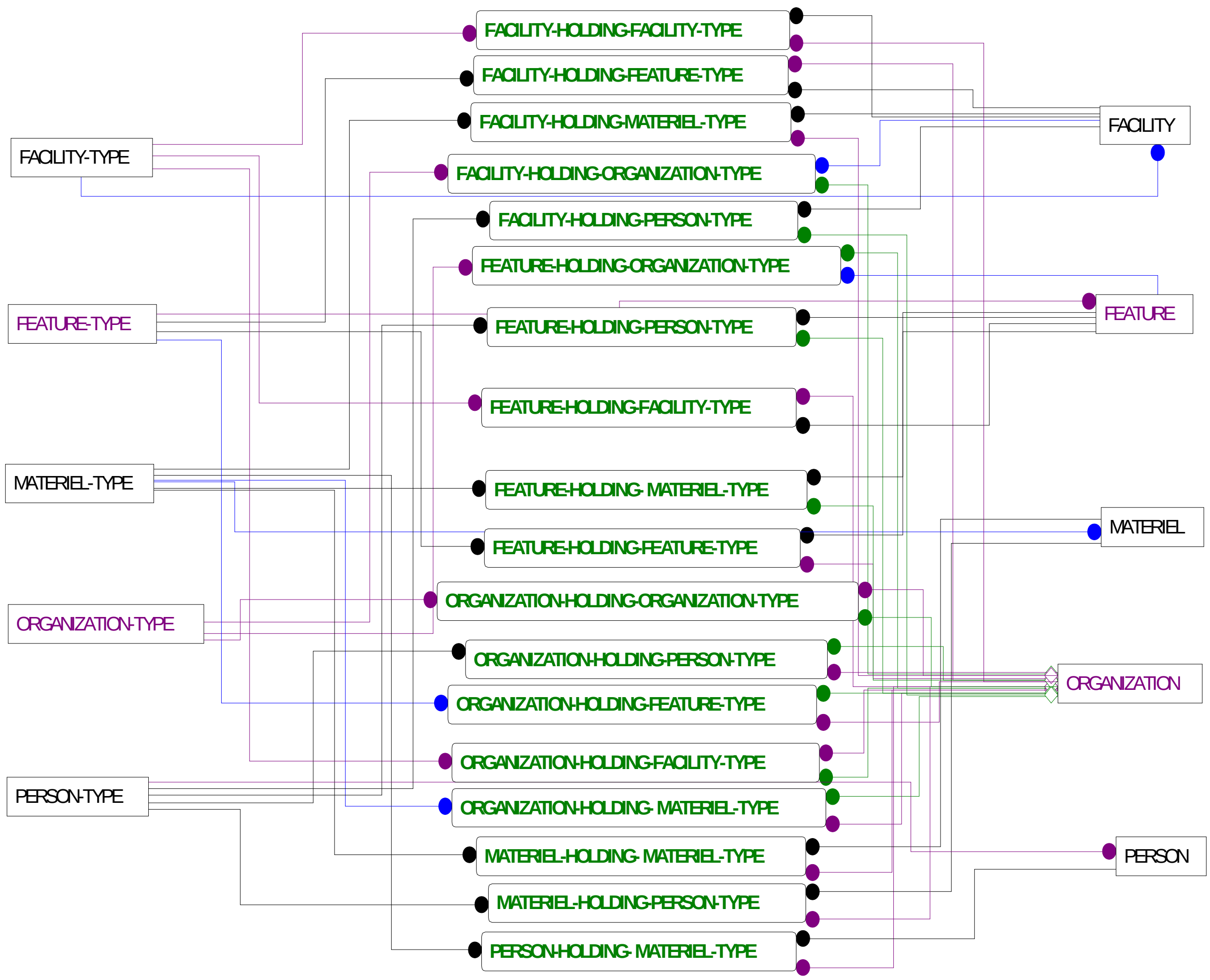


Figure E.4.2.1-1. AICDM Structures for HOLDINGs

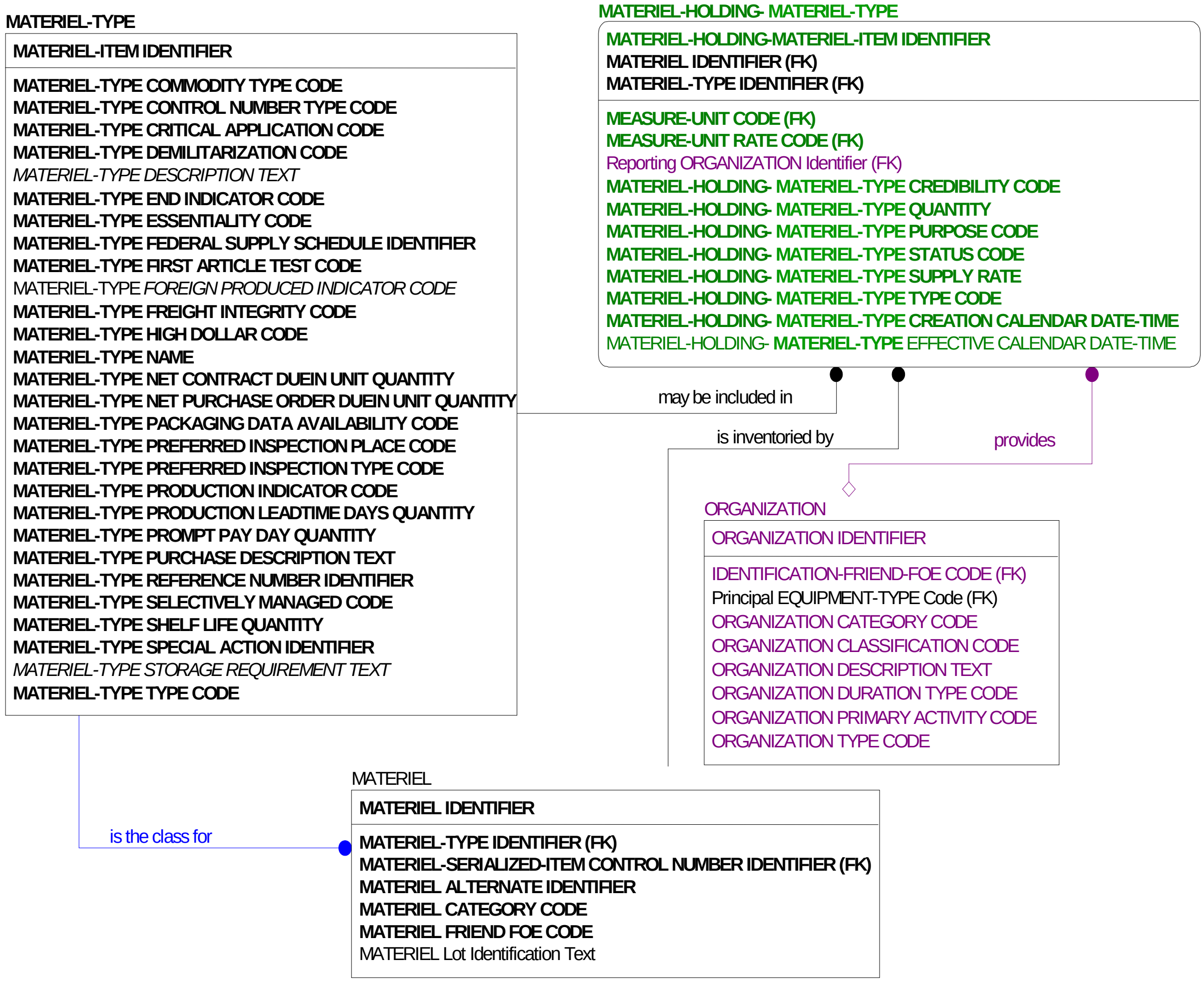


Figure E.4.2.1-2. AICDM Holding Structure for MATERIEL

Although there is no REPORTING-DATA in the AICDM, Figure E.4.2.1-2 shows that the attributes defined for holdings are able to track the quantities of any given MATERIEL-TYPE held by an instance of MATERIEL, and through the STATUS CODE can capture information about the condition of the holdings (e.g., ready, operational, out of action), in basically the same way as the LC2IEDM. In addition, the PURPOSE CODE permits the specification of what each instance of the holding is to be used for (e.g., operational, simulation), and the CREDIBILITY CODE gives an indication as to how trustworthy the particular estimate is (e.g., trusted, suspect).

E.5. Representation Of Authorized Lists Of Resources And Actual Inventories in the JCDB

The Joint Common Database (JCDB) is the Army's current implementation of a C2CDM-compliant information system aimed at supporting key ABCS systems. Due to the nature of the DoD standardization program, as well as the presence of Army specific data requirements, not all data structures in the JCDB match the current specifications contained in the DoD Data

Architecture. However, the Army is vigorously pursuing the inclusion of these new data requirements into the set of DoD-wide data standards, with a view to ensure maximum interoperability with all the other Services and Agencies.

E.5.1 Establishments

The JCDB does not contain explicit structures for capturing the concept of establishments as they exist in the C2CDM, the AICDM, the CADM and the LC2IEDM. If we consider establishments as a special kind of associations among types of battlefield objects, we could claim that the JCDB possesses a more general construct for linking instances of MATERIEL-TYPE with other instances of the same entity, namely MATERIEL-TYPE-ASSOCIATION. This entity could potentially be used to capture some of the establishment information for MATERIEL-TYPE provided a new attribute for capturing the quantity of each class is created therein. This would then provide a capability similar to the MATERIEL-TYPE-ESTABLISHMENT-MATERIEL-TYPE-DETAIL in the AICDM. Figure E.5.1-1 shows the data structure as it currently exists in JCDB version 4.4.

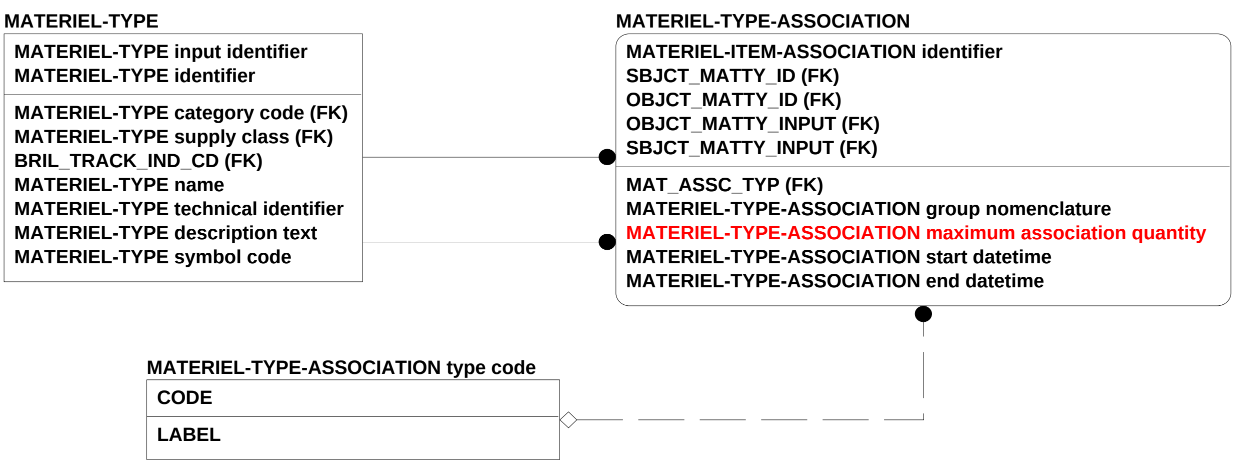


Figure E.5.1-1. MATERIEL-TYPE-ASSOCIATION Structure in JCDB

There are no similar associative structures for FACILITY-TYPE, FEATURE-TYPE, or ORGANIZATION-TYPE. Therefore, neither the data requirements for FEATURE-TYPE-ESTABLISHMENT-FEATURE-TYPE-DETAIL, nor FACILITY-TYPE-ESTABLISHMENT-FACILITY-TYPE-DETAIL, nor ORGANIZATION-TYPE-ESTABLISHMENT-ORGANIZATION-TYPE-DETAIL can be satisfied with the current JCDB. In addition, neither can any of the other establishments be represented.

E.5.2 Holdings

The JCDB contains a series of data structures for Holdings that more closely resemble the structures of C2CDM and the other models mentioned in the previous section. It should be noted, however, that whereas C2CDM and AICDM provide 18 entities to capture the various types of holdings, the JCDB only lists 4 holding entities. In addition, the JCDB has chosen to create some new kinds of holdings specifically aimed at capturing information related to enemy ORGANIZATION, enemy PERSON, and enemy MATERIEL. As Figure E.5.2-1 shows, there are two more HOLDINGS associated with the new entities.

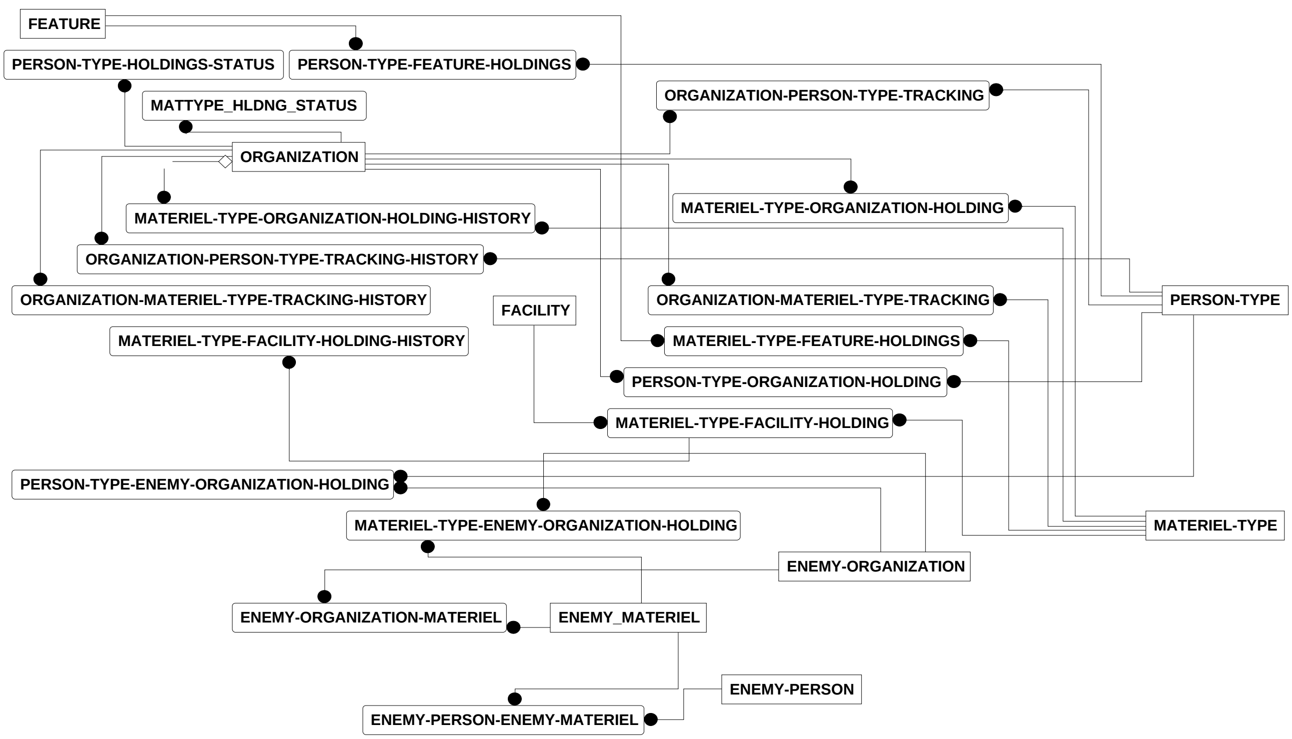


Figure E.5.2-1. JCDB Holding Structures

E.6. Representation of Authorized Lists of Resources and Actual Inventories in the CLDM

At present the structures that constitute the CLDM, i.e., the totality of the entities, attributes and relationships that comprise the “view” of the DDA labeled DUSD(L) or Logistics, do not include any of the structures required for C2 to specify either the authorized lists of resources or the actual on-hand quantities that one may wish to associate with any given battlefield object. This is not surprising given that the CLDM scope and objective was not to provide information for conducting C2 operations, but for tracking materiel classes and their procurement, storage and shipment to the military units who requisition them.

Because the CLDM and the AICDM both are built with standardized data structures and already share a substantial number of the key entities, namely MATERIEL, MATERIEL-TYPE, ORGANIZATION, PERSON, PERSON-TYPE, FACILITY, and FACILITY-TYPE, it would be relatively easy to include the remaining entities needed to provide a capability similar to the AICDM to handle establishments and holdings, in case this is needed.[68]

E.7. Representation of Authorized Lists of Resources and Actual Inventories in the LIDB

The current version of the LIDB Logical Data Model does not contain any of the Establishment and Holding entities needed for capturing information related to them. As in the case of the CLDM, addition of the necessary structures could be achieved given that the model already contains the following entities, namely, FACILITY, FACILITY-TYPE, ORGANIZATION, ORGANIZATION-TYPE, PERSON and PERSON-TYPE. These entities map readily to those used in the DDA for handling establishments and holdings.

It should be noted, however, that the LIDB does not contain an entity for MATERIEL-TYPE, and that the current MATERIEL, defined as "A MATERIEL PLAN", is modeled as a subtype of PROJECT (defined as "A PLAN"), which in itself may also complicate things because it would be redundant with the DDA entity PLAN.

E.8. Representation of Authorized Lists of Resources and Actual Inventories in the CADM

E.8.1 Establishments

The CADM is a logical data model primarily geared to support the data requirements specified in the *C4ISR Architecture Framework, Version 2.0*. It contains only three of the five types of battlefield objects present in the C2CDM and the AICDM, namely, FACILITY-TYPE, MATERIEL-TYPE and ORGANIZATION-TYPE. As Figures E.8.1-1 and E.8.1-2 show, the CADM supports two types of establishments, one for ORGANIZATION-TYPE and the other for MATERIEL-TYPE. Army extensions have been proposed aimed at supporting the description of TOEs and MTOEs, as they currently exist in the C4RDP database.

Because the CADM does not currently support personnel requirements, it may require further extensions to support that aspect of the establishments requirements.

[68] As part of the deliverables for the task under which this report has been created there will be a separate analysis on the alternative ways in which both the AICDM and CLDM can be "integrated" so that the C2 as well as the Logistics data requirements can be better supported in the overall DoD Data Architecture data model.

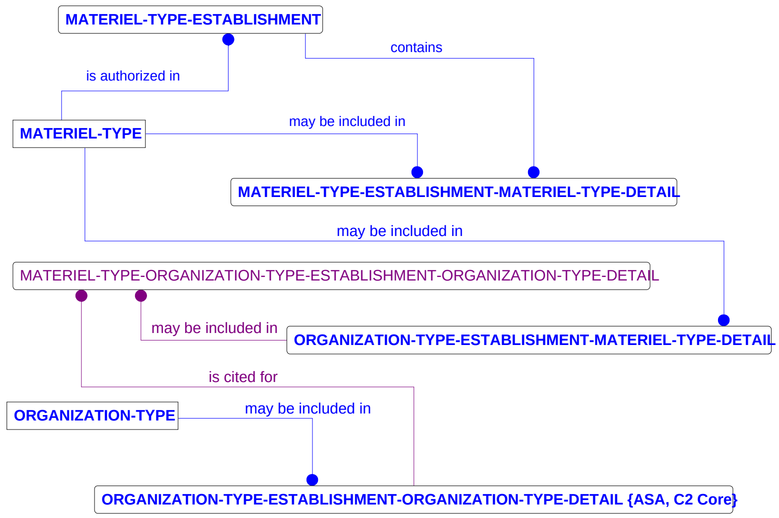


Figure E.8.1-1. CADM MATERIEL ESTABLISHMENT Structures

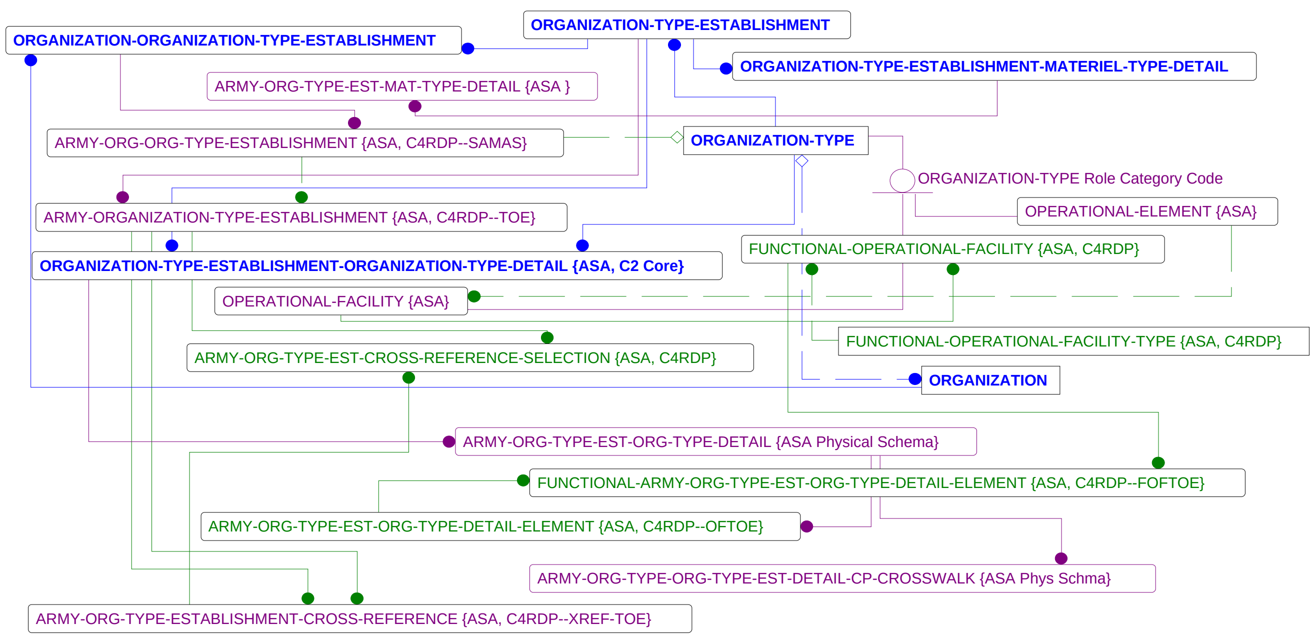


Figure E.8.1-2. CADM Organization Establishment Structures

E.8.2 Holdings

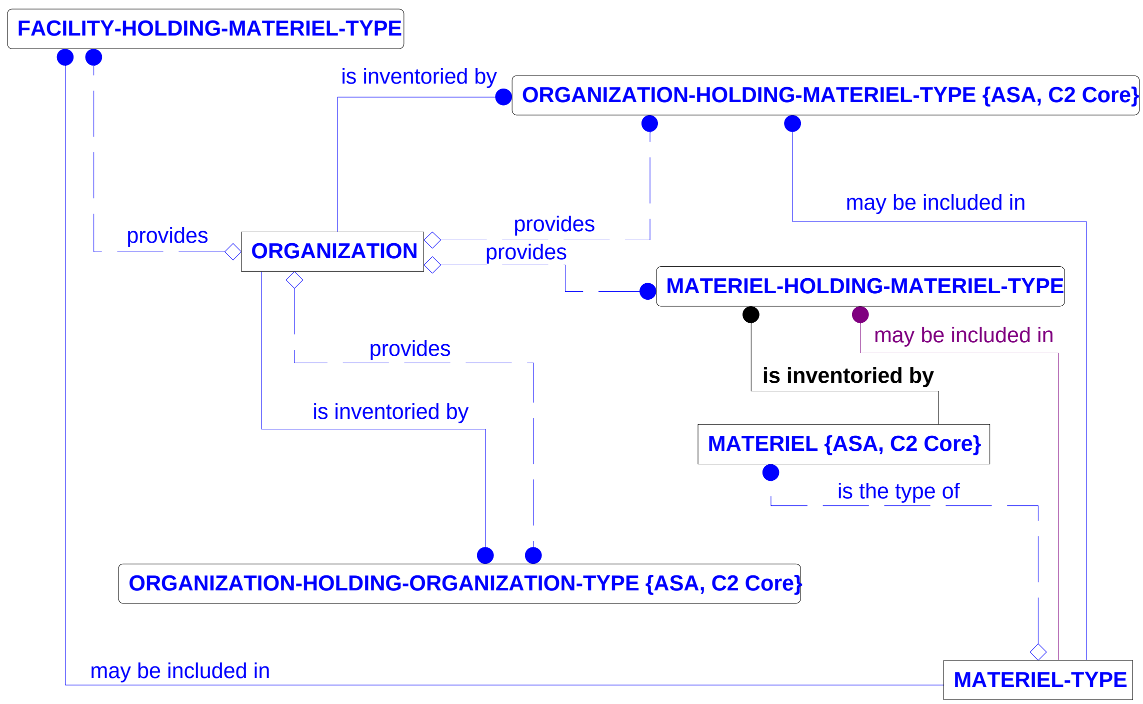


Figure E.8.2-1. CADM Holdings Structures

The CADM supports four types of Holdings: (1) FACILITY-HOLDING-MATERIEL-TYPE, (2) ORGANIZATION-HOLDING-MATERIEL-TYPE, (3) MATERIEL-HOLDING-MATERIEL-TYPE, and (4) ORGANIZATION-HOLDING-ORGANIZATION-TYPE. Figure E.8.2-1 shows the CADM holding structures, which are basically identical to those contained in the C2CDM and in the AICDM.

E.9 Summary and Conclusions

Table E.9-1 shows schematically the findings of the analysis conducted in this Appendix for the six data models that potentially could play a role in the Materiel domain, either as the point of departure for a tightly integrated Logistics-C2 data representation and standardization, or as sources and users of reference, stationary and dynamic MATERIEL data. As has been indicated elsewhere in this document, the specification of the data structures contained in a data model are a reflection of the scope of the model, the nature of the operations that the model is meant to support, and the information exchange requirements, if any, that are imposed on it. In the majority of cases integrating two or more data models is a very complex task, which requires extensive restructuring of the original constructs. In the Materiel domain area, however, it is fair to say that all the models can trace their origins to either the DDA, the C2CDM, or some version of the precursor model, the Generic Hub. As the remarks made in each section show, the situation for the majority of the models considered is one of omission of portions of the corresponding parallel structures for HOLDINGs and ESTABLISHMENTs, rather than a completely different approach for handling the Materiel domain data. Even in the cases where no explicit use of the HOLDINGs and ESTABLISHMENTs currently exist, the basic entities needed to support them are present, or can be added from the stock of standardized structures that reside in the DDA.

Table E.9-1. Summary of HOLDINGs and ESTABLISHMENTs Supported by the 6 Data Models Analyzed

STRUCTURES SUPPORTED	LC2IEDM	AICDM	JCDB	CLDM	CADM	LIDB
FACILITY-HOLDINGs	all[69]	all	1	no	1	no
FEATURE-HOLDINGs	all	all	2	no	no	no
MATERIEL-HOLDINGs	all	2	no	no	1	no
ORGANIZATION-HOLDINGs	all	all	2	no	2	no
PERSON-HOLDINGs	all	1	no	no	no	no
FACILITY-ESTABLISHMENTs	all	all	no	no	no	no
FEATURE-ESTABLISHMENTs	all	2	no	no	no	no
MATERIEL-ESTABLISHMENTs	all	2	no	no	2	no
ORGANIZATION-ESTABLISHMENTs	all	4	no	no	2	no
PERSON-ESTABLISHMENTs	all	1	no	no	no	no

Even though the analysis presented in this Appendix has focused only on a specific aspect of the Materiel domain, it is probably fair to say that of all the modeling strategies used, the one adopted in the LC2IEDM is the most economical and powerful due to its simplicity and level of abstraction. As a result, it is strongly recommended that the entities OBJECT-TYPE and OBJECT-ITEM be seriously considered for use in the Materiel domain and Organization domain, not only because they permit substantial simplifications in the models, but more importantly, because they would provide the natural focus for the introduction of Enterprise Identifiers when implementing the respective model-compliant databases. In other words, both the OBJECT-TYPE Identifier and the OBJECT-ITEM Identifier as used in the LC2IEDM could readily support the introduction of EIDs with minimal or no structural impact.

[69] Given that there are 5 battlefield object items and 5 battlefield object types, the maximum number of HOLDINGs is 25 (or five for each of the five battlefield object items), and the maximum number of ESTABLISHMENTs is also 25 (or five for each of the five battlefield object types). Some of these constructs may not have real operational meaning (PERSONs may not have holdings of control features). In LC2IEDM restrictions may be accomplished via business rules, but in principle all HOLDINGs and ESTABLISHMENTs are possible. In AICDM the choice has been made to explicitly state which of the HOLDINGs and ESTABLISHMENTs are supported. A similar approach has also been adopted in JCDB, and CLDM.

APPENDIX F

ISSUES PERTAINING TO A PREFERRED ONTOLOGY AND TAXONOMY FOR THE MATERIEL DOMAIN

APPENDIX F – ISSUES PERTAINING TO A PREFERRED ONTOLOGY AND TAXONOMY FOR THE MATERIEL DOMAIN

Introduction

The purpose of this Appendix is to discuss at a more technical level some of the implications and implementation opportunities that the adoption of EIDs in the Materiel domain could have for the classification and management of Logistics data, i.e., the taxonomies and ontologies that could be used in this data domain. Thus, even though the introduction of Enterprise Identifiers (EIDs) could make it possible to address with minimal or no ambiguity any record that exists in any of the participating enterprise databases, it is nevertheless important to model the large sets of Logistics reference and stationary data in a manner that facilitates their understanding, and reflects the particular user community's needs, e.g., C2, Modeling and Simulation, Logistics.

To that effect, it is customary to aggregate the data in terms of the "objects" that constitute the main focus of the enterprise, and to categorize them along hierarchical structures that permit their separation into mutually exclusive classes. Loosely speaking, an "ontology" addresses the universe of objects to be considered in the enterprise, or, alternatively, it defines the universe of things that can be mentioned in an interface. "Taxonomy," on the other hand, deals with the orderly and systematic classification of the members of any given "ontology" and supports among other things the normalization of data when modeled within relational databases.

At present, the portion of the DoD Data Architecture (DDA) that supports Logistics, namely the Corporate Logistics Data Model (CLDM), as well as other data models used by the Army, such as the Joint Common Data Base (JCDB), contain extensive ontologies that reflect their main business practices. On the other hand, the taxonomies that have been proposed for the Materiel Domain in the majority of these models suffer from substantial inconsistencies, and arguably may not be sufficient to support all the needs of the DoD enterprise.

F.1 Comparison of Current Ontologies in the Materiel Domain

F.1.1 The Materiel Domain Ontology and Taxonomy in JCDB

Table F.1.1-1 shows the ontology for the Materiel domain that exists in the JCDB, and Figure F.1.1-1 depicts its associated taxonomy.

Table F.1.1-1. Ontology for the Materiel Domain in JCDB

Entity Name	Definition
AIRCRAFT-TYPE	A classification of a device designed to be capable of atmospheric flight.
CONSUMABLE-MATERIEL-TYPE	A MATERIEL-TYPE which can be expended.
EQUIPMENT-TYPE	A categorization of MATERIEL-TYPE that provides capability through repeated use.
MATERIEL	An object of interest that is non-human, mobile, and physical.
MATERIEL-TYPE	A characterization of a MATERIEL asset.
MATERIEL-TYPE-ASSOCIATION	The association of one specific MATERIEL-TYPE to another specific MATERIEL-TYPE.
MUNITION	A MATERIEL-TYPE that is employed to inflict damage for a military purpose.
NETWORK-BRIDGE	A device which forwards traffic between network segments based on data link layer information. These segments would have a common network layer address.
NETWORK-CABLE	The wires or fibers, or groups of, usually surrounded by protective shell, that can carry voice, data, or video for a specific NETWORK..
NETWORK-DEVICE	A device used to implement a NETWORK..
NETWORK-GATEWAY	A protocol converter. A router is a layer 3 (network layer) gateway, and a mail gateway is a layer 7 (application layer) gateway. (NOTE: This term was previously used for a router or other kind of internetworking device but this use is now deprecated.)
NETWORK-HUB	A network device connected to several other devices. NOTE: A hub is used to connect several computers together. In a message handling service, a number of local computers might exchange messages solely with a hub computer. The hub would be responsible for exchanging messages with other hubs and non-local computers.
NETWORK-RADIO	A device used to transmit and/or receive electromagnetic emissions for a specific NETWORK.
NETWORK-ROUTER	A device which forwards packets between networks. The forwarding decision is based on network layer information and routing tables, often constructed by routing protocols.
POWER-GENERATION-UNIT	An equipment used to generate, and initiate the distribution of, POWER.
SENSOR-TYPE	A classification of equipment which detects objects and activities.
SHELTER	A MATERIEL-TYPE for transport and/or housing of MATERIEL.
SOFTWARE-APPLICATION	A complete, self-contained software program that performs a specific function directly for the user. [This is in contrast to system software such as the operating system kernel, server processes and libraries which exists to support application programs.] Editors, spreadsheets, and text formatters are common examples of applications. Network applications include clients such as those for FTP, electronic mail and telnet.
VEHICLE-TYPE	A classification of a VEHICLE.
WEAPON-TYPE	A classification of an instrument used in offensive or defensive combat to disarm, persuade, or defeat an enemy.

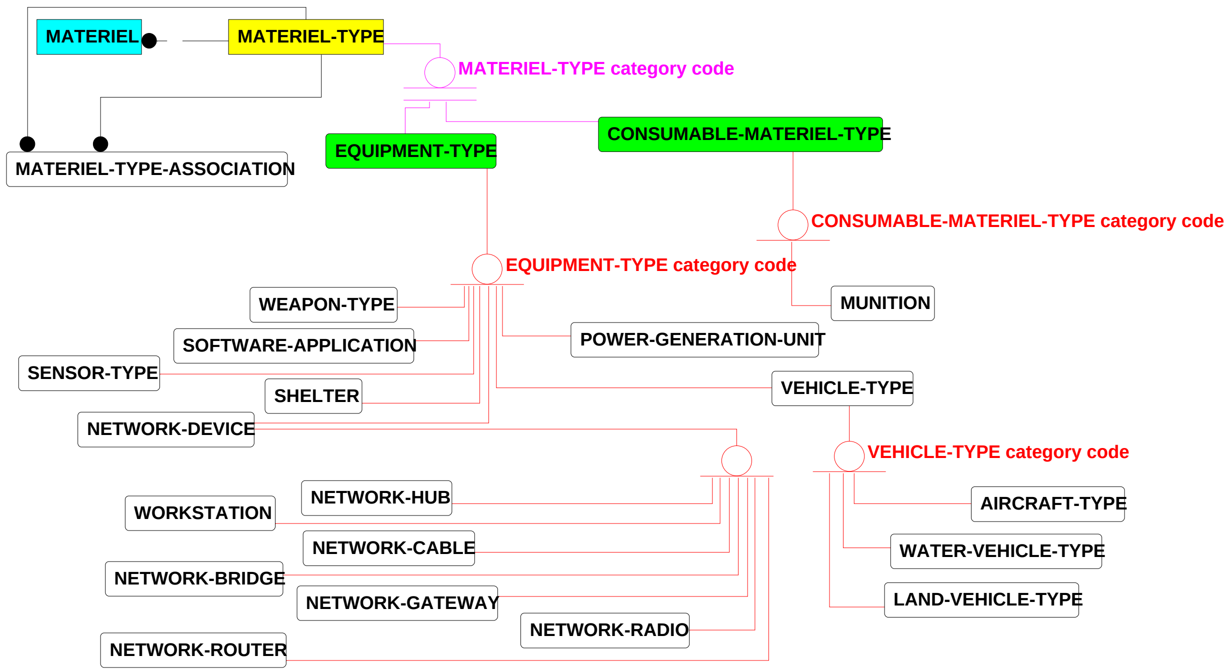


Figure F.1.1-1. Taxonomy for the Materiel Domain in JCDB

Inspection of the JCDB taxonomy shows that only one class of consumables, namely MUNITION, is at present tracked in this model, leaving out other types of consumables that arguably may be pertinent to the prosecution of battle, e.g., fuel, medical supplies. The classes under EQUIPMENT cover only a portion of the overall spectrum of what can be categorized as EQUIPMENT. Neither SYSTEMs, nor the concept of PLATFORM, are considered as part of the Materiel domain of the JCDB. This may constitute a shortfall when interfacing systems based on the JCDB with, for example, either systems based on the C4ISR Core Architecture Data Model (CADM) that collect system-related information, or simulation systems that use PLATFORM to depict the integration of units, weapons and personnel.

F.1.2 The Materiel Domain Ontology and Taxonomy in CLDM

Table F.1.2-1 shows the core ontology for the Materiel domain that exists in the CLDM, i.e., the entities that directly pertain to MATERIEL-TYPE in terms of its subclasses, as opposed to those that, for example, may be required for its production, such as engineering drawings.

A brief review of the entries listed in **Table F.1.2-1** shows that the CLDM ontology is heavily biased towards abstract classifications that serve the cataloguing and procurement of materiel within a Logistics enterprise. No functional labels are apparent, as was the case in JCDB. Analysis of the domain values for the classes of materiel that are catalogued at the federal level show substantial overlap with other entities that are shown outside the MATERIEL-TYPE hierarchy. Thus, for example, FUEL-TYPEs, VEHICLE-TYPEs, AIRCRAFT-TYPEs and SHIP-TYPEs are all part of the values that the MATERIEL-TYPE-SUPPLY NATIONAL STOCK NUMBER IDENTIFIER covers, and are also separate entities in their own right. In addition, some objects appear classified according to whether an organization (such as the Federal Government) or a person has some claim of ownership to it or not, instead of being classified according to their inherent properties (e.g., GOVERNMENT-VEHICLE, PRIVATELY-OWNED-VEHICLE). In terms of the taxonomy adequate to serve Logistics, at a minimum the CLDM should show either (non-identifying) relationships, or associative tables among these entities and MATERIEL-TYPE. This would facilitate the exchange of information when using one or the other portion of the model. Ideally, all these entities should be harmonized within broader taxonomies.

Table F.1.2-1. Ontology for the Materiel Domain in CLDM

Entity Name	Definition
AIRCRAFT	(6) (A) A DEVICE DESIGNED TO BE CAPABLE OF ATMOSPHERIC FLIGHT.
AIRCRAFT-TYPE	(7) (A) A CLASSIFICATION OF A DEVICE DESIGNED TO BE CAPABLE OF ATMOSPHERIC FLIGHT.
DOCUMENT	(119) (A) RECORDED INFORMATION REGARDLESS OF PHYSICAL FORM.
DOCUMENT-MATERIEL-TYPE	(8686) (A) A DOCUMENT MANAGED AS A MATERIEL-TYPE .
END-ITEM	(7703) (A) A PORTION OF AN ACQUISITION-PROGRAM EXPLICITLY IDENTIFIED FOR MANAGEMENT REPORTING PURPOSES.
END-MATERIEL-TYPE	(12595) (A) A COMPLEX MATERIEL-TYPE DESIGNATED AS A SELF-CONTAINED OBJECT.
END-MATERIEL-TYPE-END-ITEM-ASSOCIATION	(12596) (A) THE ASSOCIATION OF AN END-ITEM AND AN END-MATERIEL-TYPE .
EQUPMENT-DESIGNATION	(14302) (A) A DESIGNATION FOR EQUIPMENT MANAGED AS AN END-ITEM.
FUEL	(342) (A) A SUBSTANCE USED TO GENERATE ENERGY.
FUEL-TYPE	(2094) (A) A CATEGORY OF SUPPLY USED TO PRODUCE HEAT OR POWER BY BURNING.
GOVERNMENT-NON-SELF-PROPELLED-EQUIPMENT	(11675) (A) A CATEGORY OF MATERIEL-SERIALIZED-ASSET THAT CANNOT MOVE BY ITSELF.
GOVERNMENT-VEHICLE	(2298) (A) A PIECE OF MECHANIZED EQUIPMENT THAT IS OWNED BY THE FEDERAL GOVERNMENT THAT CAN BE UTILIZED FOR TRANSPORTING PASSENGERS, GOODS OR APPARATUS
LAND-VEHICLE-TYPE	(1160) (A) THE CLASSIFICATION OF A TRANSPORTATION DEVICE THAT OPERATES ON LAND.
MAJOR-EQUIPMENT-TYPE	(8) (A) A CATEGORY OF EQUIPMENT THAT IS OF PRIMARY MILITARY SIGNIFICANCE.
MATERIEL	(337) (A) AN OBJECT OF INTEREST THAT IS NON-HUMAN, MOBILE, AND PHYSICAL.
MATERIEL-SERIALIZED-ITEM	(783) (A) A MATERIEL RECOGNIZED INDIVIDUALLY BY A UNIQUELY ASSIGNED IDENTIFYING SYMBOLIC DESIGNATION.
MATERIEL-TYPE-ALLOWANCE	(7249) (A) A SET OF MATERIEL-TYPE PERMITTED FOR AN ORGANIZATION.
MATERIEL-TYPE-ALLOWANCE-LINE-ITEM	(7250) (A) A MATERIEL-TYPE PERMITTED FOR A MATERIEL-ALLOWANCE.
MATERIEL-TYPE-ASSET	(2480) (A) A MATERIEL-TYPE TO WHICH THE DEPARTMENT OF DEFENSE HAS LEGAL CLAIM OF PROPRIETORSHIP.
MATERIEL-TYPE-DOCUMENT	(788) (A) A COLLECTION OF COMPLEX, COMPOUND OR COMPLEX AND COMPOUND MATERIEL-TYPE DATA, INDEPENDENT OF PHYSICAL OR STRUCTURAL, FORMAT, WHICH MAY BE SUBJECT TO DISTRIBUTION.
MATERIEL-TYPE-DOCUMENT-MEDIA-TYPE	(8511) (A) THE KIND OF MATERIAL UTILIZED FOR A PUBLICATION.
MATERIEL-TYPE-DUE-IN	(752) (A) MATERIEL-TYPE EXPECTED TO BE RECEIVED.

Entity Name	Definition
MATERIEL-TYPE-SERIALIZED-ASSET	(2485) (A) A MATERIEL-TYPE THAT IS RECOGNIZED INDIVIDUALLY BY A UNIQUELY ASSIGNED SYMBOLIC DESIGNATION TO WHICH THE DEPARTMENT OF DEFENSE HAS LEGAL CLAIM OF PROPRIETORSHIP.
MATERIEL-TYPE-SERIALIZED-ASSET-SAMPLE	(14236) (A) A SAMPLE THAT IS A MATERIEL-TYPE-SERIALIZED-ASSET.
MATERIEL-TYPE-SERIALIZED-ASSET-TYPE	(15276) (A) A KIND OF MATERIEL-TYPE-SERIALIZED-ASSET.
MATERIEL-TYPE	(787) (A) A CHARACTERIZATION OF A MATERIEL-TYPE-ASSET.
MATERIEL-TYPE-ASSOCIATION	(780) (A) AN ASSOCIATION OF ONE SPECIFIC MATERIEL-TYPE TO ANOTHER SPECIFIC MATERIEL-TYPE.
MATERIEL-TYPE-AXLE-CHARACTERISTIC	(8790) (A) A CATEGORY OF MATERIEL-TYPE-CHARACTERISTIC WHICH ARE FEATURES OF AN AXLE.
MATERIEL-TYPE-CHARACTERISTIC	(8789) (A) A FEATURE OF A MATERIEL-TYPE .
MATERIEL-TYPE-CHARACTERISTIC-MEASUREMENT	(8787) (A) A MEASURE OF A MATERIEL-TYPE-CHARACTERISTIC.
MATERIEL-TYPE-FOREIGN-PRODUCED	(11494) (A) A MATERIEL-TYPE-PRODUCTION THAT IS PRODUCED IN A COUNTRY OTHER THAN THE UNITED STATES.
MATERIEL-TYPE-MANAGEMENT-CATEGORY	(817) (A) THE HIGHEST LEVEL OF COMMODITY CLASSIFICATION TO IDENTIFY LIKE MATERIEL-TYPES.
MATERIEL-TYPE-MANAGEMENT-FEDERAL-SUPPLY-CLASS	(826) (A) THE SECOND LEVEL CLASSIFICATION FOR GROUPING LIKE MATERIEL-TYPES.
MATERIEL-TYPE-MANAGEMENT-FEDERAL-SUPPLY-CLASS-GROUP-ASSOCIATION	(13532) (A) THE ASSOCIATION OF A FEDERAL-SUPPLY-CLASS-MATERIEL-MANAGEMENT-GROUP WITH A MATERIEL-TYPE-MANAGEMENT-FEDERAL-SUPPLY-CLASS.
MATERIEL-TYPE-PRECIOUS-METAL	(11456) (A) AN ASSOCIATION BETWEEN A MATERIEL-TYPE AND A PRECIOUS-METAL.
MATERIEL-TYPE-PRODUCTION	(733) (A) A MATERIEL-TYPE THAT IS IDENTIFIED BY PRODUCER OR INDUSTRY MANUFACTURER.
MATERIEL-TYPE-PRODUCTION-LOT	(11499) (A) A GROUP OF MATERIEL-TYPE-PRODUCTION MANUFACTURED AND CONTROLLED UNDER THE SAME SET OF SPECIFICATIONS.
MATERIEL-TYPE-PRODUCTION-LOT-BATCH	(13238) (A) A SUB GROUPING OF MATERIEL-TYPE-PRODUCTION-LOT PRODUCED UNDER MORE EXACTING PRODUCTION SPECIFICATIONS.
MATERIEL-TYPE-PRODUCTION-LOT-SAMPLE	(14234) (A) A SAMPLE WHOSE SOURCE IS IDENTIFIED AS A MEMBER OF A MATERIEL-TYPE-PRODUCTION-LOT.
MATERIEL-TYPE-SUPPLY	(732) (A) A MATERIEL-TYPE- ASSET THAT HAS BEEN CATALOGED.
MATERIEL-TYPE-SUPPLY-GROUP	(11458) (A) AN ASSOCIATION OF A MATERIEL-TYPE-SUPPLY WITH A MATERIEL-MANAGEMENT-GROUP.
MATERIEL-TYPE-SUPPLY-GROUP-CHANGE	(11459) (A) A MODIFICATION TO A MATERIEL-TYPE-SUPPLY-GROUP.
MATERIEL-TYPE-SUPPLY-PRODUCTION	(11461) (A) AN ASSOCIATION BETWEEN A MATERIEL-TYPE-SUPPLY AND A MATERIEL-TYPE-PRODUCTION.
PRECIOUS-METAL	(11468) (A) A SOLID CHEMICAL ELEMENT WITH HIGH INTRINSIC VALUE.
PRIVATELY-OWNED-VEHICLE	(2301) (A) A TRANSPORTATION DEVICE THAT IS OWNED BY A PRIVATE CITIZEN.
SHIP	(24) (A) A MOVEABLE CRAFT DESIGNED FOR NAVIGATION IN OR ON THE WATER.
SHIP-TYPE	(26) (A) A CATEGORY OF A VESSEL BUILT FOR DEEP WATER NAVIGATION.
STANDARD-OPERATION	(4281) (A) AN ESTABLISHED PROCEDURE FOR ACCOMPLISHING A UNIT OF WORK.

Entity Name	Definition
STORAGE-AREA	(7006) (A) A PHYSICAL SPACE WITHIN A FACILITY THAT IS DESIGNATED FOR THE STORAGE OF MA TERIEL.
SUPPLY-CONDITION	(2486) (A) THE QUALITATIVE STATUS OF A MATERIEL-TYPE .
SUPPLY-ITEM-CONTROL	(829) (A) THE MANAGEMENT CONTROL EXERCISED ON A MATERIEL-TYPE-SUPPLY FOR A SPECIFIC USER.
SUPPLY-ITEM-REPLACEMENT	(784) (A) A MATERIEL-TYPE-SUPPLY THAT SUPERSEDES ANOTHER MATERIEL-TYPE-SUPPLY.
SUPPLY-ITEM-SUBSTITUTE	(795) (A) AN ALTERNATE MATERIEL-TYPE-SUPPLY.
SUPPLY-MATERIEL-TYPE	(9813) (A) A MATERIEL-TYPE THAT IS MANAGED IN THE FEDERAL SUPPLY SYSTEM.
VEHICLE-TYPE	(46) (A) A CLASSIFICATION OF A VEHICLE.

F.1.3 General Issues Concerning Taxonomies

Before addressing possible alternatives for the Materiel domain taxonomies in paragraph F.1.4 below, it is important to highlight the conceptual difficulties that are inherent in the development of any taxonomy in general, and of a given taxonomy in particular. This is necessary for the Materiel domain as it presently exists in the DDA, as well as any of the other models that the Army has developed.

F.1.3.1 Remarks on Formal Treatments of Taxonomies

It is neither within the scope of this document nor possible in the space of this Appendix to treat the theoretical underpinnings of what is involved in the development of a taxonomy. However, it should be noted that various aspects of this endeavor have been looked at in the context of Applied Logic, particularly in areas such as Mereology, which is broadly defined as the logic of parts and wholes.

We understand a "taxonomy" for the purpose of managing data within a database as an instantiation of a series of statements of the type "α is a subtype of β", " or equivalently "α is a β", such that if there is a series of classes α, β, γ, . . . in a given domain D, they are all disjoint. Furthermore, there is always a class which can be conceived as the sum of all the classes α, β, γ, . . . for the domain D. It, therefore, becomes apparent that "generalizations" and "subtype hierarchies", as they are specified in IDEF1X, map directly to this view of "taxonomy" we just introduced, because within a given hierarchy the "subtypes" are all disjoint, and the supertype always includes all the instances of the subtypes.

Aside from the elementary results derived within the theory of mereology that support formal identity between two classes and the like, it is important to know that the relationship "is a subtype of" or equivalently "is a" is transitive (i.e., for every α, for every β, and for every γ, if α is subtype of β, and β is a subtype of γ, then α is a subtype of γ). Furthermore, any item is completely determined by its parts; items are identical when they have the same parts in common; and, in principle, any two classes whatsoever may be summed up (i.e., made into subtypes of a more general class).

Let's assume that the essential equivalence holds between the basic results of this branch of applied logic and the way in which "taxonomies" are built. Then, since "subtype hierarchies" in IDEF1X are defined just as in Mereology (that is as breaking their respective domains into disjoint classes), it may be impossible to construct a single MATERIEL-TYPE taxonomy that can readily accommodate all possible classifications based on functionality, use, nature of the business, etc. This is because none of these approaches lends itself to a decomposition into cleanly disjoint classes. In other words, except for a MATERIEL-TYPE taxonomy based on a strict decomposition of the instances of MATERIEL-TYPE into their constituent parts (themselves instances of MATERIEL-TYPE), it may not be possible to construct subtype hierarchies that are both:

1) stable and conceptually consistent across the whole enterprise, and
2) based on the type of functionality, or the envisioned use of the instances of MATERIEL-TYPE being classified thereby.

Thus, if something like a PLATFORM is subtyped into LAND-PLATFORM, AIR-PLATFORM and WATER-PLATFORM (which is a taxonomy obviously based on the functionality of this class of MATERIEL-TYPE), then the introduction of an amphibious PLATFORM at some later time may require either creating yet another explicit class (such as AMPHIBIOUS-PLATFORM), or requiring its instances be treated implicitly only within the supertype PLATFORM. If the former, then the hierarchy will have to be changed (violating the requirement for stability), which may impact a fair number of systems based on the previous version which did not include the new class. If the latter, then data that needs to be collected about the new class may not have a place holder in any of the data structures that comprise the database scheme (making the taxonomy less useful for practical purposes).

On the other hand, taxonomies are based on the relationship "is a subtype of", which is transitive, as was shown above. This enables databases using them to instantiate "inheritance" at the attribute level, and supports "data normalization". Thus, data values that apply to the supertype need not be repeated at the subtype level. This greatly simplifies the management of data, and prevents inconsistencies and data corruption as a result of record updates. Last but not least, hierarchies allow developers to clarify the meaning of the data and to achieve greater specificity at the logical level.

From the above, it is safe to say that:

1) A universal taxonomy for a domain as broad as the Materiel domain is quite unlikely to emerge, given that:
    a) most taxonomies are built using "functionality" and "use" criteria, rather than a straight decomposition into constituent components of the various classes of MATERIEL-TYPE, and
    b) such criteria do not lend themselves to decompositions into disjoint classes.
2) Localized taxonomies, where simple extensions of the enumerated classes are sufficient to accommodate the requirements of the communities that choose to exchange data, are quite feasible and desirable.

3) "Taxonomies," based on criteria other than the one that classifies all instances of MATERIEL-TYPE according to its constituent components, may not be capable of supporting all the data requirements of the enterprise. Nevertheless, they are quite useful for:

    a) achieving interoperability among selected communities, and
    b) designing efficient databases with a high degree of normalization

The following section, therefore, assumes that the taxonomies discussed are not meant as general taxonomies for the totality of the Materiel domain (and for all the user communities that interface with this domain), but simply as alternative approaches for supporting some aspects of data interoperability between the C2 and the Logistics community.

F.1.4 A Proposed Materiel Domain Ontology and Taxonomy

As shown above, both the CLDM and the JCDB have deficiencies which make their use outside of their domains difficult. A possible solution to this may consist in the adoption of a new taxonomy that permits the current classes to be readily accommodated, while providing other classes. While these other classes have been identified as required by communities such as Modeling and Simulation, at present they cannot be supported by either of the models considered. Figure F.1.3.2-1 shows a possible high-level taxonomy for MATERIEL-TYPE. A comparison with the JCDB entities specified for the Materiel domain above already shows substantial overlap (i.e., SENSOR-TYPE, NETWORK-DEVICE-TYPE and WEAPON-TYPE), exists in both taxonomies.

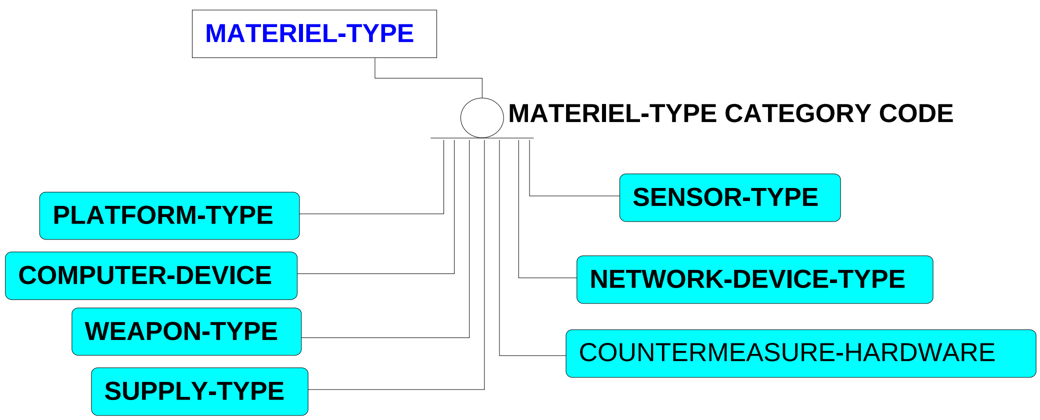


Figure F.1.4-1. Alternative High-Level Taxonomy for the Materiel Domain

Each one of the sub-type entities depicted in Figure F.1.4-1 in turn can be further decomposed to accommodate the ontologies that either JCDB, CLDM or any other model may have. For example, we could envision creating a generalization under the SUPPLY-TYPE entity as shown in Figure F.1.4-2 to capture not only MUNITION but FUEL-TYPE, as well as instances of FOOD and MEDICAL-SUPPLY.

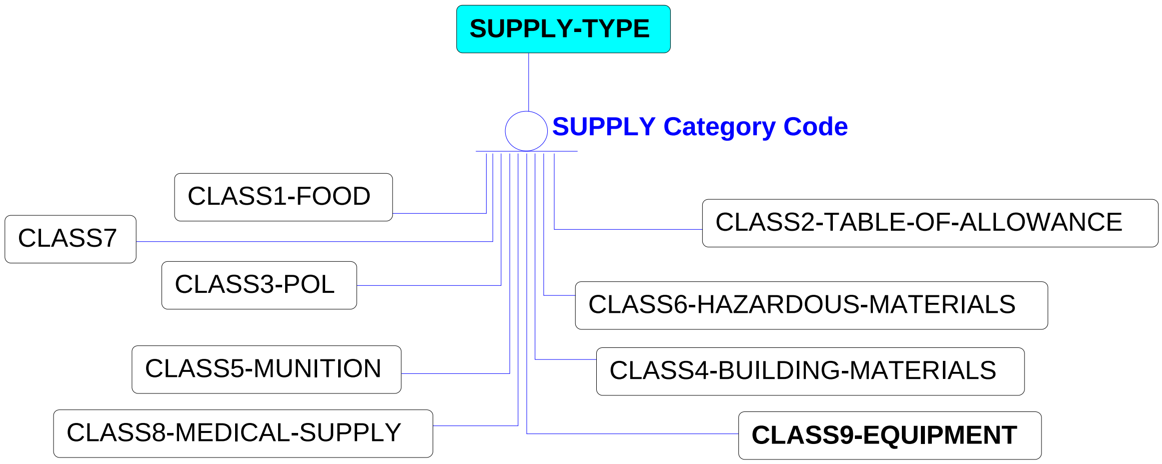


Figure F.1.4-2. Alternative Generalization for SUPPLY-TYPE

The JCDB entities SHELTER, SOFTWARE-APPLICATION, and POWER-GENERATION-UNIT could arguably be also modeled as part of CLASS9-EQUIPMENT, as shown in Figure F.1.4-3.

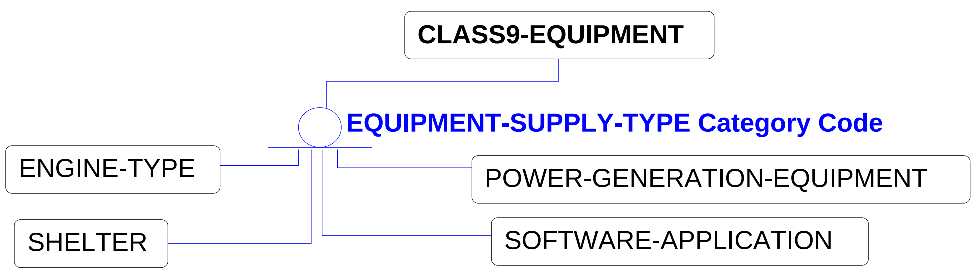


Figure F.1.4-3. Alternative Generalization for EQUIPMENT-TYPE

Similarly, VEHICLE-TYPE, AIRCRAFT-TYPE, and SHIP-TYPE, could be brought under the concept of PLATFORM-TYPE, as it is shown in Figure F.1.4-4. In addition, other types of platforms, which at present are not captured by either the JCDB or the CLDM, could be accommodated as well, such as SPACE-PLATFORMs.

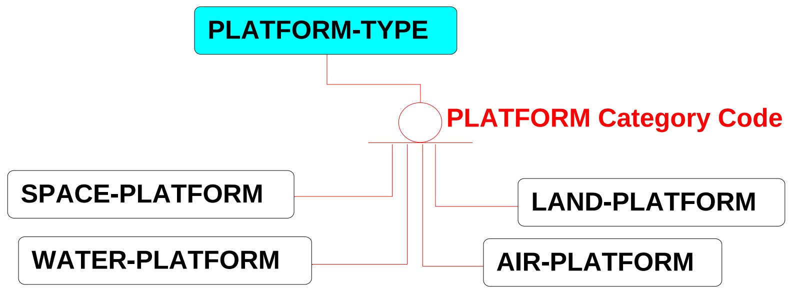


Figure F.1.4-4. Alternative Generalization for PLATFORM-TYPE

Another possible generalization concerns the hierarchy under WEAPON-TYPE. A preliminary, and by no means exhaustive, set of entities to consider as subtypes is shown in Figure F.1.4-5.

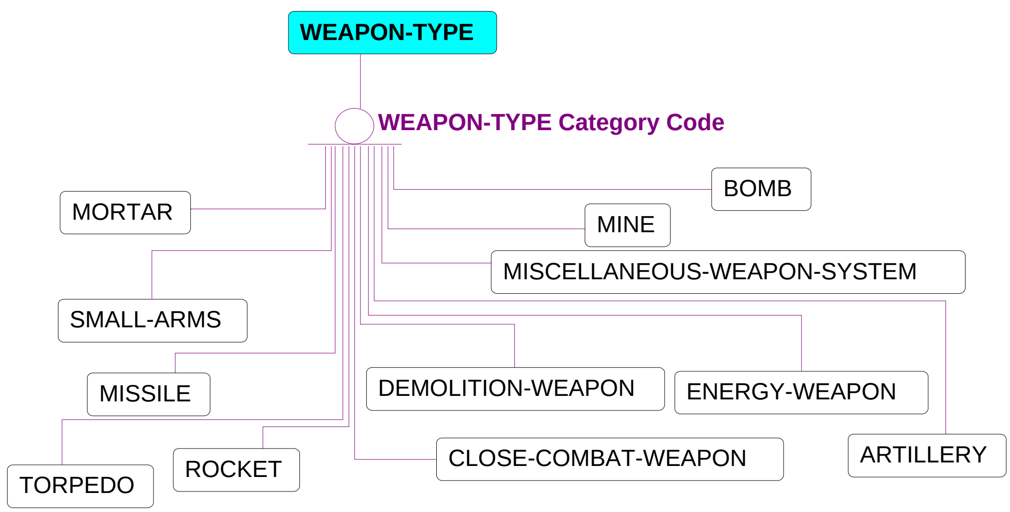


Figure F.1.4-5. Alternative Generalization for WEAPON-TYPE

Finally, it should be noted that some generalizations that currently exist may be better handled via enumerated domains, rather than through the introduction of new entities. This applies in particular to those portions of the ontology for the Materiel domain that pertain to highly dynamic technologies, such as computers and communications.

APPENDIX G

MODELING AND IMPLEMENTATION IMPLICATIONS ARISING FROM THE USE OF EIDs

APPENDIX G – MODELING AND IMPLEMENTATION IMPLICATIONS ARISING FROM THE USE OF EIDs

G.1 Introduction

The use of enterprise identifiers (EIDs) presents an interesting issue in the relational database environment. As was mentioned in the main body of this document, as well as in Appendix D, "A Computer Database and Logical Data Model Primer," EIDs are surrogate keys, and, therefore, are composed of only a single (database) field whose value is not derived from any other entity. This means that if an EID is a primary key, then the primary key is a single attribute that is not inherited from any other entity. In other words, all the records in that table can be identified via the EID attribute assigned to that table. We refer to this situation by saying that the entity is independent. It is, therefore, true to say that if all primary keys in all the tables of a given database are EIDs, then all the tables in that database can be implemented as independent entities. The consequences of this interpretation for implementation purposes are many. In this appendix two cases will be discussed in some detail, namely, the implications for: (a) associative entities, and (b) child entities.

G.2 Modeling and Implementation Implications for Associative Entities

If we were to consistently replace with EIDs all the key attributes of an existing database, or those of a planned one, then it would be perfectly acceptable to represent all the tables thereof as independent ones. But since relational databases depend strictly on the migration of keys to re-link the data that has been broken into its atomic constituents, the question arises as to how best to recapture this "relational" feature. This question is particularly relevant when dealing with associative entities, which currently inherit the keys of both parents, and, when required, introduce a third key to allow for multiple associations of the same pair of instances. In other words, what are the implications of changing this type of entities into independent ones? The answer is actually quite simple. The new independent entity could carry the two keys from the associative pair of parents as before, but only to perform the original lookup or join operations, and not to uniquely identify the records in the associative entity, as is currently the case.

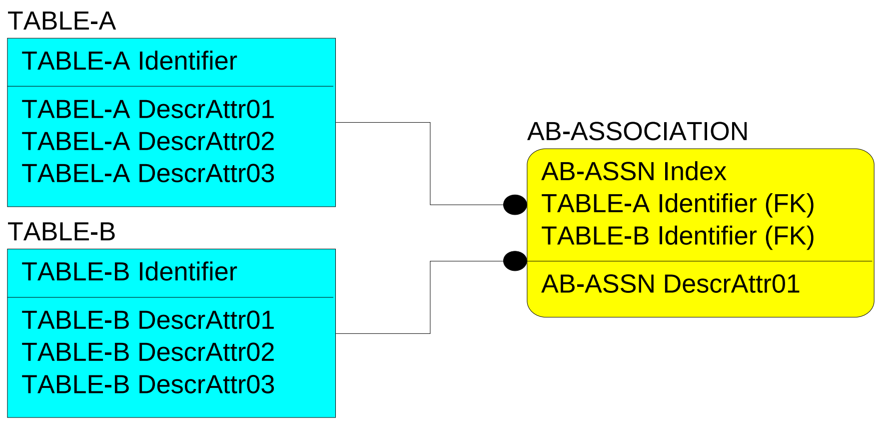


Figure G.2-1. Current Modeling of Associative Entities without EIDs

Figure G.2-1 depicts the current modeling approach used for associative entities. In the notional case shown therein, two entities, generically labeled TABLE-A and TABLE-B, have been linked via the AB-ASSOCIATION table. For simplicity, only one identifier has been chosen for each of the parent entities, namely, TABLE-A Identifier and TABLE-B Identifier.

As shown in the Figure G.2-1, the associative entity AB-ASSOCIATION then inherits both keys from the parent tables as foreign keys, meaning that its records will be identified with the help of these keys. A third attribute, namely AB-ASSN Index, has also been included as part of the key for all records in the AB-ASSOCIATION table, because instances of TABLE-A and TABLE-B could appear more than once in the AB-ASSOCIATION table. As examples, the AB-ASSOCIATION table may represent the relationship between an instance of ORGANIZATION and an instance of PERSON; and ORGANIZATION may perform various functions in regard to the PERSON, e.g., "trains", "houses", "deploys".

Note that in the current RDBMS implementations where no requirement for global uniqueness of the identifiers is imposed on the databases, it is perfectly acceptable to have the actual values of TABLE-A Identifier, TABLE-B Identifier, and AB-ASSN Index be the same. In other words, there is nothing that prevents assigning the value "23322" to all three of these attributes under the current implementation of RDBMS engines. The only restriction imposed is that within TABLE-A, and TABLE-B there be no "repeats" for either the TABLE-A Identifier, or the TABLE-B Identifier. For the AB-ASSOCIATION table the only restriction is that the combination of TABLE-A Identifier, TABLE-B Identifier, and AB-ASSN Index be unique. Thus, entries such as TABLE-A Identifier = "23322", TABLE-A Identifier = "23322", AB-ASSN Index = "23322" would be perfectly valid, and any of the commercially available RDBMS engines would only reject attempts at creating a second record with the same combination. But as long as any of the three entries is different, all commercially available RDBMS engines would not stop the record from being added to the AB-ASSOCIATION table.

Note also that the currently available RDBMS engines do not impose any restriction on the structure of the key. Thus, it would be perfectly acceptable to have TABLE-A Identifier = "33323322", TABLE-B Identifier = "ACTG-23322" and AB-ASSN Index = "099". In other words, the data types and data lengths of the key attributes in the current databases can be anything the implementer wants, as long as they do not repeat within any given table.

If one were to take the construct shown in Figure G.2-1 and implement EIDs, which by definition are "globally unique" and hence can unequivocally identify any record in any table of all the participating databases of the enterprise, then associations such as the one schematically shown in Figure G.2-1 above could be "simplified" in terms of the key structure. To discuss the most generic of cases, it will be assumed that the changes will be done to a legacy database, where thousands of records already exist and are routinely queried via the key structures shown in Figure G.2-1.

The introduction of EIDs could be accomplished in either of two ways: (a) insertion of a new EID attribute in every table with the intent to use it as the primary key for that table, and corresponding demotion of the existing key structure to the role of alternate keys or inversion entries, or (b) insertion in every table of a new EID attribute in the role of an alternate key or an inversion entry, and retention of the existing key structure for record identification.

Figure G.2-2 shows the first alternative (a) where the existing key structure is modified completely, albeit none of the previous key "information" is destroyed. Because every record of every table in all of the participating databases of the enterprise will have a distinct, globally unique value via its EID, introducing this approach to model associations among the entities now permits the tagging of every record in TABLE-A, TABLE-B and AB-ASSOCIATION with its own EID. Therefore, the situation discussed above would never arise, i.e., there would be no possibility of finding that TABLE-A EID, TABLE-B EID, and AB-ASSN EID have the same value[70].

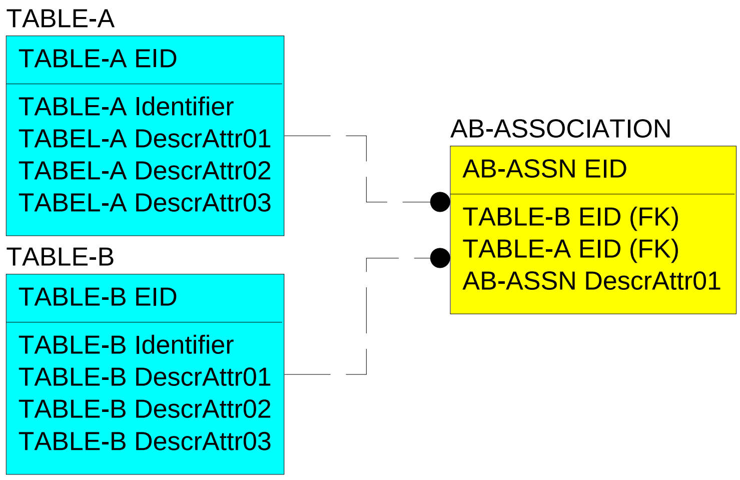


Figure G.2-2. First Alternative Modeling of Associative Entities Using EIDs

[70] Note that when dealing with legacy systems the original keys are demoted and not simply deleted because they are likely to be the "business keys" of the tables, and preserving them allows for legacy SQL queries to still occur.

There are three advantages of alternative (a) as shown in Figure G.2-2. First, there is potential for simplification of SQL queries, particularly in the case where the associative entity has been set up to allow for updates in one or more of its descriptive attributes. Once the EID for the particular record that belongs to the relationship between the two parents is known, all queries can be written using that value, instead of having to use two or more keys for retrieving that record. Second, intelligent and efficient software can be written that takes advantage of the uniqueness of the keys across the whole enterprise. This would permit us to implement distributed queries that treat all the participating databases in the enterprise as a single virtual database, and "joins" could be executed that presently would be very inefficient or impossible. Third, this structure facilitates the implementation of true "plug and play" applications based on a key structure that is common throughout the whole enterprise.

The other alternative (b) for implementing EIDs in a legacy database is shown in Figure G.2-3. The structure of the database is modified only by the addition of another attribute, the globally unique EID, which, as indicated by the notation, can serve as an alternate key. The advantages of this approach are (1) minimal changes to the key structures of the legacy system, and, therefore, minimal changes required to the current SQL queries; (2) potential for migration towards a system that eventually replaces the current key structure with the new one based on EIDs; and (3) potential for writing new software based on the presence of EIDs in all the tables of the participating databases of the enterprise that complements and enhances the legacy software.

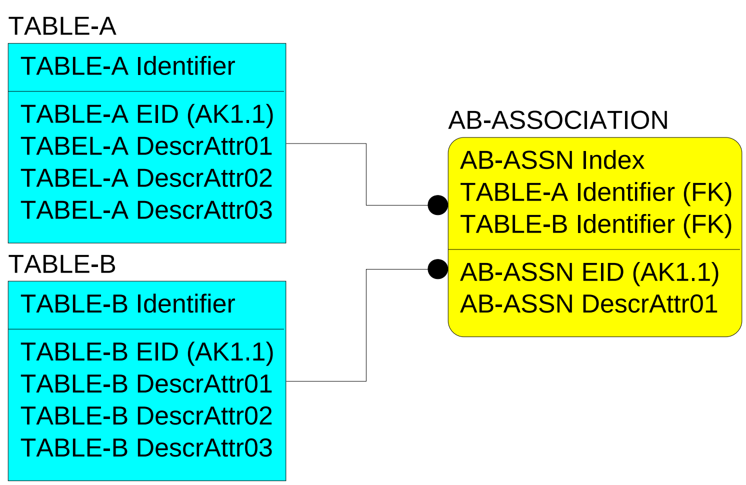


Figure G.2-3. Second Alternative Modeling of Associative Entities Using EIDs

G.3 Modeling and Implementation Implications for Child Entities

The current implementation of child entities in relational databases is shown in Figure G.3-1. In this case an entity shown as TABLE-C has a child entity called TABLE-D. All the records in TABLE-C are identified via the key attribute TABLE-C Identifier. As discussed above, the only restriction that all commercially available RDBMS engines impose on such attributes is that they be unique within the table where they exist. The key structure of the child entity, TABLE-D, shows that all its records require the value of the TABLE-C Identifier. In addition, to implement the "many" cardinality of the relationship depicted, we also need a second attribute, namely TABLE-D Identifier, to form a composite unique key.

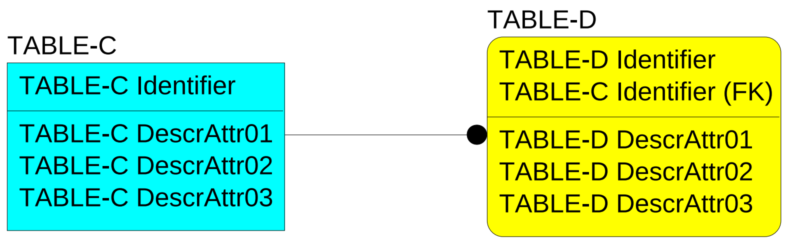


Figure G.3-1. Current Modeling of Child Entities without using EIDs

As was the case when we discussed associative entities, we could introduce EIDs as full replacements for the current key structure. This first alternative is schematically depicted in Figure G.3-2 below.

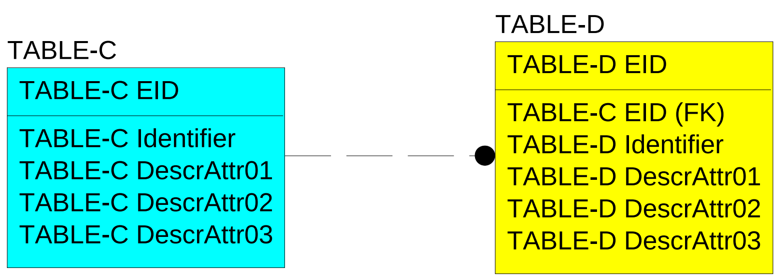


Figure G.3-2. First Alternative Modeling of Child Entities using EIDs

In this case, the pre-existing key structure is made into part of the descriptive attributes of each of the entities, and each record is now uniquely identified via the globally unique values of its own EID. The use of this approach benefits in the same way as the implementation of EIDs in modeling associative entities. Just as we could introduce the EIDs in the associative entities without any major changes to the current database structure, so too can it be done for the parent-child situation.

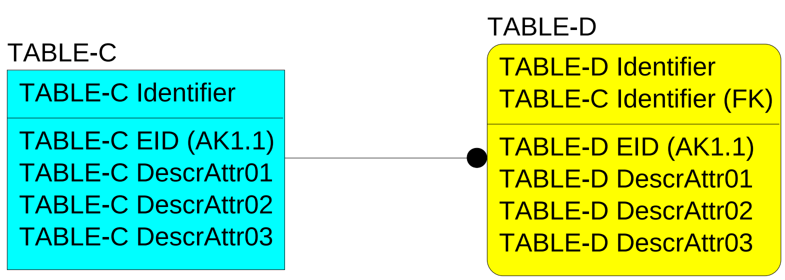


Figure G.3-3. Second Alternative Modeling of Child Entities using EIDs

Figure G.3-3 schematically depicts a second alternative. It shows how globally unique EIDs can be implemented while retaining the traditional structure for modeling child entities. As discussed in the previous section, the new attributes are used as alternate keys, giving implementers another way to write applications that can run not only in the local environment, but can be generalized to perform distributed queries as well.

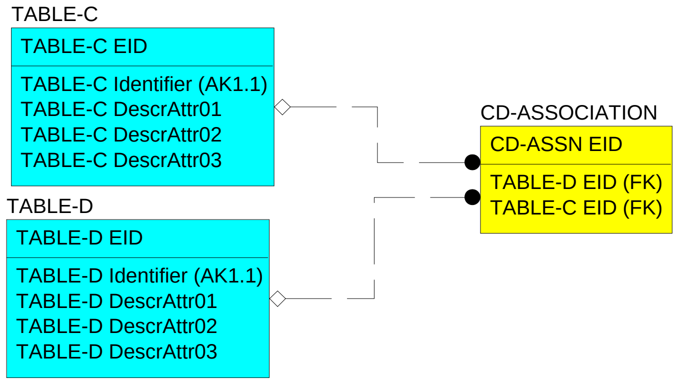


Figure G.3-4. Recasting Child Entities as Associative Entities using EIDs

A third alternative for implementing EIDs in the context of child entities is to recast them as associative entities. Figure G.3-4 shows how this alternative would look.

The advantages of this approach are in essence the same as those discussed above for the use of EIDs in the context of associative entities. There would be, however, an increase in the number of tables needed to capture the data. On the other hand, substantial reduction in the size of records could be achieved since the values, that potentially get repeated hundred of thousands of times in child entities, would be replaced by a single record consisting of three 64-bit surrogate keys in the new table.

G.4 Generalizations Hierarchies Using EIDs

One of the great advantages that data models such as the LC2IEDM have, is that they already use high-level generalizations for handling all their major objects. Figures G.4-1 and G.4-2 show the two generalizations that handle the battlefield objects defined in the LC2IEDM, namely, FACILITY, FEATURE, MATERIEL, ORGANIZATION, PERSON, and their corresponding types, FACILITY-TYPE, FEATURE-TYPE, MATERIEL-TYPE, ORGANIZATION-TYPE, and PERSON-TYPE. Clearly, adopting this type of approach makes the introduction of EIDs very easy, since it is already the case that all the battlefield objects and their types will share within each hierarchy the same key structure. Therefore, changing both the OBJECT-ITEM Identifier and the OBJECT-TYPE Identifier into EIDs would percolate down each chain with minimal or no modeling impact.

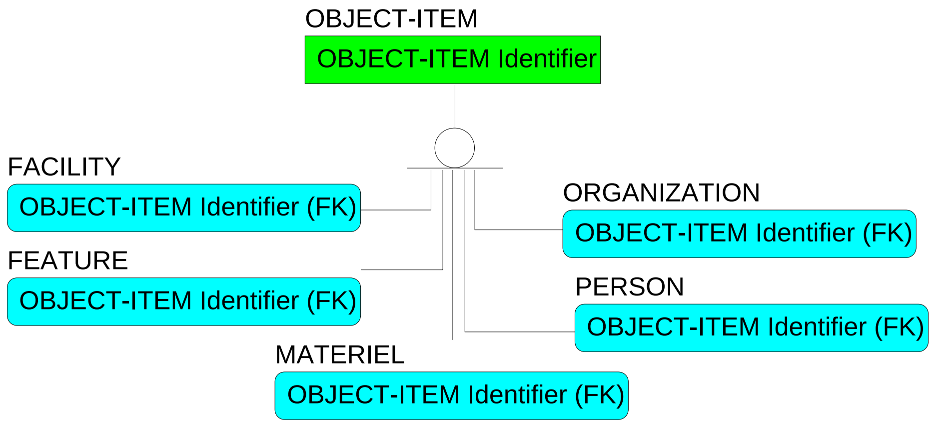


Figure G.4-1. High-Level Generalization for Battlefield Objects

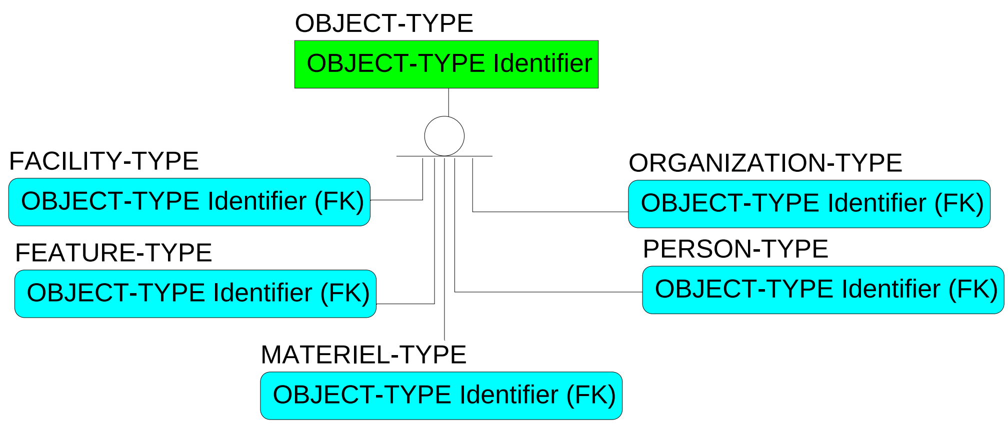


Figure G.4-2. High-Level Generalization for Battlefield Object-Types

A further simplification of the key structure at the logical data model level could be achieved if we were to postulate a generalization for OBJECT-ITEM and OBJECT-TYPE, as is shown in Figure G.4-3. Changing this OBJECT Identifier into a globally unique EID, namely, GENERALIZED OBJECT Identifier, would effectively guarantee that all the battlefield objects are managed in exactly the same manner, and that no two records in any of those tables can be referred to by the same value of the key attribute.

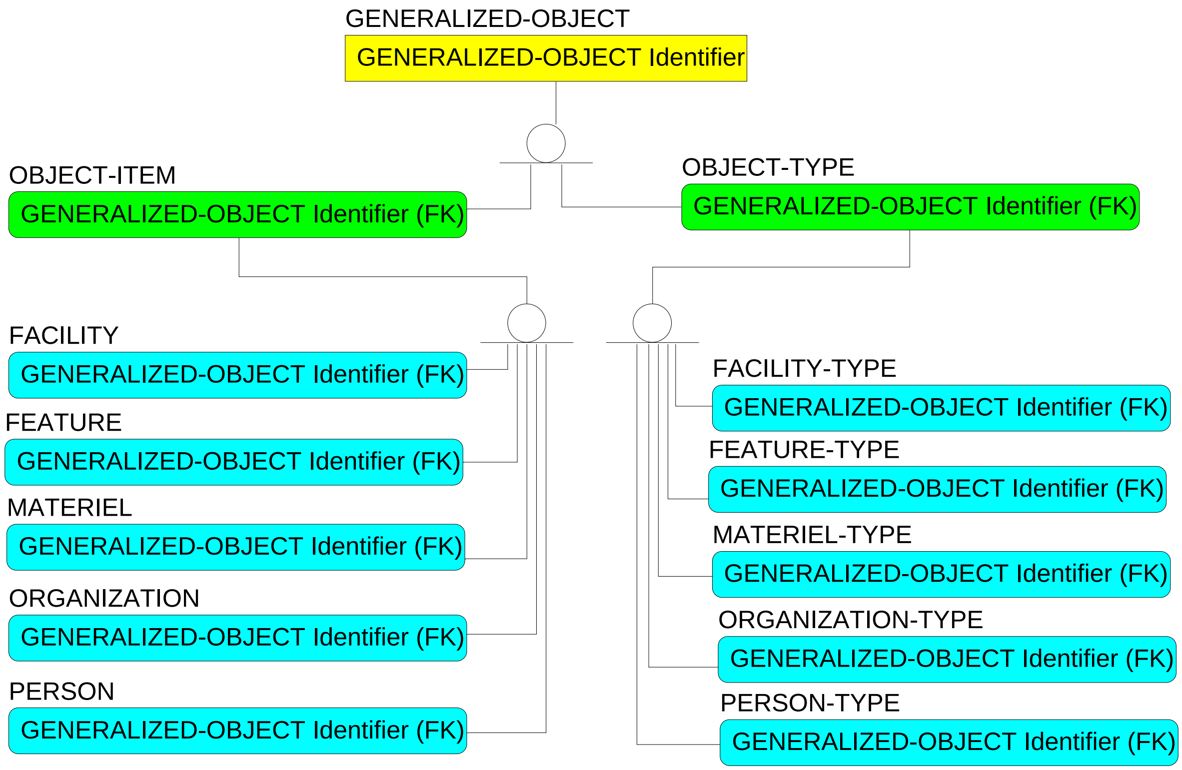


Figure G.4-3. A Generalization for OBJECT-ITEM and OBJECT-TYPE

Further, if one were to include an associative entity, namely, GENOBJECT-ASSOCIATION, to enable the linkage between any pair of instances of GENERALIZED-OBJECT, then much of the logical data model could be reduced at implementation time to two relatively large tables. One table would contain "things"—either items or types—of interest, and the other table would define how they are related to each other. Such a modeling approach is depicted in Figure G.4-4.

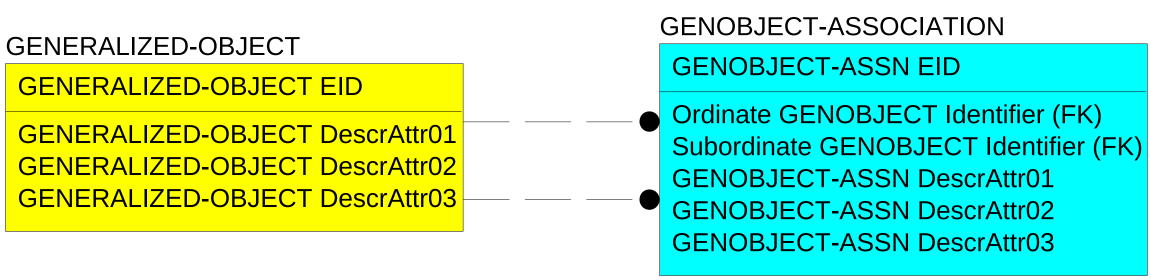


Figure G.4-4. Simplification of Key and Entity Structures Using EIDs

The points illustrated above hint at the fact that the adoption of a database architecture whereby all records in all tables of all participating databases of the enterprise derive their keys in a manner that ensures that no key value is ever duplicated is conceptually equivalent to a generalization that subsumes all the members of the enterprise ontology under a small number of supertype entities[71]. Given that most of the information in relational databases resides in the relationships that are established after data is carefully normalized, it would appear that the logical consequence of this approach permits subsuming all these associations into a single relationships entity.

It should be noted that setting up a system in this fashion may not be easier than using the traditional approach. Further, for the purpose of data requirements specification, keeping the entities separate facilitates the analysis and ensures that not only database engineers, but functional experts and end users, can participate in the design and help refine the product.

G.5 Conclusions on the Implementation Implications for Associative and Child Entities using EIDs

Database designers and engineers may object to this policy because it limits some shortcuts in the query process and can be construed as requiring unnecessarily more space, i.e., one more attribute for keys in each table than under the ordinary approach.

Additionally, careful modeling must be performed when replacing the old key structure with the new EIDs, especially where business rules have been captured through them. A prime example of this is the so-called "intentional unification of keys". In the current approach, where the child entity inherits the identifying key of the parent, it is possible to use this to create other structures that are related both to the child entity and the parent, and to ensure via the intentional unification of their keys that the values in the new tables restrict themselves to those which exist in the child entity. The use of EIDs demotes the keys of the parent to non-identifying attributes in the child table, and does not permit the keys in the first child to continue migrating down the path of other child entities. Hence, EIDs cannot take advantage of the intentional unification of keys.

However, this is a small price to pay compared to the complete and unencumbered flexibility and expandability provided by EIDs. This topic is covered thoroughly in the articles listed in footnote [21] of the main body. Further, many of the objections presented about EIDs disregard the fact that most queries and database operations are not executed by humans, but by sophisticated applications programs that can exploit the features of EIDs to significantly improve database performance and reduce overhead.

---

[71] A generalization depicts a single semantic object, and, therefore, uses the key of the supertype for all the records of the subtypes. That is why the introduction of EIDs does not affect the modeling of subtype hierarchies in the way that it may affect the modeling of associative and child entities as shown in the preceding sections. On the other hand, as discussed in this section, the use of supertypes constitutes an ideal entry point for the introduction of EIDs. Adoption of OBJECT-TYPE and OBJECT-ITEM, as already modeled in LC2IEDM, coupled with the use of EIDs for both of these supertypes could permit the implementation of many of the advantages associated with their use and would have minimal impact on the database structure per se.

APPENDIX H

SEARCH OF PROFESSIONAL LITERATURE
ON ENTERPRISE KEYS

APPENDIX H – SEARCH OF PROFESSIONAL LITERATURE ON ENTERPRISE KEYS

Introduction

This appendix documents the results of the research of the professional database industry literature on enterprise surrogate keys with respect to the pros and cons for their use in distributed information systems, and any lessons learned from instances where they have in fact been introduced as replacements for existing primary keys. After an exhaustive search of web-based sources and database industry publications and journals, only one recorded instance was found where enterprise keys have been inserted as replacements for business primary keys in a large system of distributed databases. This occurred when JB Hunt Transport, Inc., an Atlanta, Georgia based nationwide trucking firm, redesigned the structures of their enterprise information systems by substituting enterprise surrogate keys for the existing business primary keys in their databases. In addition, only one set of articles authored by Dr Tom Johnston in *Data Management Review* deal directly with the issue of redesigning existing distributed information systems through the insertion of enterprise primary keys. All other references relate to the advantages of using surrogate keys as primary keys of relational database tables during the initial design and construction of databases. In this appendix, we discuss both the JB Hunt experience, Dr. Johnston's articles, and other selected references to surrogate keys in general.

H.1 JB Hunt Information Systems Redesign Project

JB Hunt Transport, Inc., is the nation's largest publicly held motor freight carrier with operations throughout all regions of the country. It controls its nationwide operations through a distributed information system consisting of multiple databases that run on an IBM DB2 relational DBMS that process more than 3 million transactions per day. The project undertaken by JB Hunt Transport, Inc., to redesign their nationwide data system is the subject of a paper published in the on-line version of the May 1998 version of *Database Programming and Design*. Mr. Mike Lonigro authored the paper titled, *The Case for the Surrogate Key*[72]. At the time, he was the senior data administrator responsible for all conceptual, logical, and physical database design at JB Hunt Transport.

As the title implies, the thrust of Lonigro's paper was to discuss the advantages of using unintelligent surrogate primary keys (which he calls "dumb" keys as opposed to "smart" keys) in lieu of smart business keys. Unfortunately, Lonigro does not describe any of the problems and solutions that were encountered during the execution of the project. However, he does state that JB Hunt has enjoyed success with the implementation garnering benefits beyond the referential integrity enhancements provided by single attribute surrogate keys. He cites two specific benefits. The first is the advantage of being able to have one consistent primary key structure (single attribute) to contend with when developing applications and interfaces to the database.

[72] Lonigro, Mike, *The Case for the Surrogate Key"*, *Intelligent Enterprise Database Programming and Design On-line,* May 1998 found on the internet at http//:www.dbpd.com/vault/9805xtra.htm

The second is increased flexibility in being able to more accurately represent real world business rules in the database such as optional relationships.

Most of Lonigro's article deals with building a convincing case as to why "dumb" surrogate primary keys are better than "smart" business primary keys in a physical database. In that regard, talks about all the benefits of surrogate keys that are usually covered by most writers on the subject such as their guaranteed uniqueness, stability, and transparency to the user. But he also makes one very important point that is not generally emphasized in most discussions on the advantages of surrogate keys.

Lonigro stresses that there is a distinct difference between the purpose of primary keys in a logical model and the purpose of primary keys in the implementation of a physical database based on the specifications of the logical model. For a logical model he states, "The logical unique key distinguishes entity occurrences from others to avoid duplicates before entry into the database. That is why this key is typically composed of attributes and relationships that the business can recognize as distinguishing occurrences logically from others." Then with regard to an actual physical database he says, "The primary key, however, is a physical design element that uniquely identifies rows after they are in the database. In fact, it is best to consider a primary key's primary purpose to be establishing relationships in the database."

The significance of this distinction is that it makes the case for using business related attributes whenever possible for the primary keys of entities in a logical model. This facilitates an accurate representation of the business rules in relationships as well as highlighting redundancies that require normalization. When a physical database designer then proceeds to design an actual database, he can insert surrogate primary keys in lieu of the logical model primary keys and re-designate the logical keys as alternate keys. However, the logical model keys will provide a clear roadmap to the designer as to what additional relationships he must create in the physical design to preserve the migration of logical foreign keys in the physical design. This addresses the point that was made at the end of Appendix D concerning the third technical question that is often raised over the use of surrogate primary keys -- only Lonigro provides a more concise and clearer explanation.

H.2 Primary Key Reengineering References

An extensive search was conducted for professional industry references dealing specifically with the subject of reengineering operational databases by converting existing business primary keys to surrogate enterprise keys. Requests were made to industry associations such as DAMA International, a professional association for Data Administrators and the Data Warehousing Institute, an industry association that promotes the development of database technology and sponsors symposiums and forums for exchanging ideas and information relating to the database industry. In addition, the Internet was also searched using a variety of keywords and phrases for any sources of information on the topic.

The results of this search did not produce any references directly relevant to the topic of reengineering with enterprise keys with the exception of the six excellent articles authored by Dr. Tom Johnston in the on-line *Data Management Review* previously cited in paragraphs 2.2.3.1 and 2.2.3.2 of this paper[73]. The remainder of this section will attempt to summarize the themes and major points made by Dr. Johnston on the subject of surrogate enterprise keys.

Dr. Tom Johnston is a consultant with a practice specializing in relational and object-oriented data architecture and data modeling across a number of industries that include banking, healthcare, and telecommunications. Dr. Johnston alluded indirectly to the paucity of published references on the subject of reengineering with enterprise keys in personal e-mail communication with one of the authors of this paper when he wrote:

> *"I have not been able to persuade any company I have worked for to use enterprise keys instead of merely surrogate keys. It's now generally accepted that intelligent keys are bad. The 'unintelligent' keys which data modelers now agree are preferable, are what they call 'surrogate keys'. But when I argue for using enterprise keys, the data modelers I work with, and their managers, aren't willing to propose that approach to their management.*
>
> *I think it's simply their failure to understand the practical value of enterprise keys over surrogate keys. Which means it's my failure to explain it adequately. The first two articles of mine you refer to are my best attempt at explaining it."*

In these two articles titled, *The Problem*, and *The Solution*, Dr. Johnston points out that the revolution in computer technology is reflected in the way that the information systems of many companies have grown into large distributed systems operating in a networked environment routinely sharing data. The result of this growth has focused attention on the inherent deficiencies of using intelligent business keys as primary keys which has caused many

---

[73] *Primary Key Reengineering Projects: The Problem*; DM Review, February, 2000; http://www.dmreview.com/master.cfm?NavID=55&EdID=1866.

Primary Key Reengineering Projects: The Solution; DM Review, March, 2000; http://www.dmreview.com/master.cfm?NavID=55&EdID=2004.

De-Embedding Foreign Keys, Part 1: DM Direct, June 2, 2000. http://www.dmreview.com/editorial/dmreview/print_action.cfm?EdID=2308.

De-Embedding Foreign Keys, Part 2; DM Direct, June 9, 2000. http://www.dmreview.com/editorial/dmreview/print_action.cfm?EdID=2322.

De-Embedding Foreign Keys, Part 3: DM Direct, June 16, 2000. http://www.dmreview.com/editorial/dmreview/print_action.cfm?EdID=2331.

De-Embedding Foreign Keys, Part 4: DM Direct, June 23, 2000. http://dmreview.com/editorial/dmreview/print_action.cfm?EdID=2341.

enterprises to undertake reengineering projects to replace intelligent keys with unintelligent keys, or surrogate keys. Unfortunately, these reengineering initiatives have been focused on maintaining entity level data integrity, i.e., enforcing the uniqueness rule on the identifiers of rows in individual tables, as opposed to enforcing uniqueness across all tables of a database or even a system of distributed databases, which Dr. Johnston calls "object integrity". The consequence of this shortcoming is a failure to correctly account for the phenomena of migrating foreign keys described in Appendix C of this paper, which Dr. Johnston calls the foreign key ripple effect. This failure eventually results in additional costs to reengineer again when the lack of object integrity begins to drive up the costs of business operations because of corrupt or inaccurate data across the information system.

With regard to the issue of the costs of reengineering, Dr. Johnston acknowledges that there is indeed a cost (there is no "free lunch"), and discusses each element of those costs such as development, one-time conversion, security, data storage, and so on. However, he contrasts those costs with the costs of doing nothing and the benefits to be accrued by reengineering correctly the first time with enterprise keys, and concludes that the benefits clearly outweigh the costs of doing nothing not to mention the costs of reengineering incorrectly. He also points out that an enterprise key within the context of relational database modeling is semantically equivalent to an object identifier as used in object oriented modeling. This is important because it may explain the reluctance of relational data modelers to embrace the concept of enterprise keys because they are accustomed to thinking about primary keys as business keys that only enforce entity level uniqueness upon individual entities and tables within a single model or database and not global uniqueness across distributed databases. So they may not fully appreciate the benefits to be derived from implementing enterprise keys into their schemas. Object oriented modelers, on the other hand, accept as mantra that an object identifier is just an unintelligent identifier whose sole purpose is to be a globally unique identifier of objects in order to unambiguously identify all objects across the enterprise environment. Thus, they are more likely to recognize enterprise keys as another type of object identifier and understand the benefits of employing enterprise keys in relational database systems.

This view was confirmed in a personal e-mail colloquy through the on-line Data Management Discussion Group with Mr. Frank Palmeri, a data administrator with the New York State Tax and Finance Department, when he wrote about a presentation that Mike Lonigro gave on his JB Hunt project. Palmeri said[74]:

> *"I heard Mike give this presentation [The Case for the Surrogate Key] at the International DB2 Users Group in Orlando last year. Mike said the big benefit of surrogate keys is ease of maintenance- he can change logical keys months after the tables are in production and not worry about affecting the physical design. He did this talk in front of a huge group of DB2 professionals, and quite frankly, I didn't think the audience gave him enough feedback, maybe because this is such a different way of doing*

[74] E-mail Index #1241, http//:www.egroups.com/group/dm-discuss/1241.html [Used by [permission of Mr. Palmeri].

things they didn't know what to ask. From Mike's article that you mentioned- 'We would do well to remember that a smart key is a dumb thing to do, while a dumb key is a smart thing to do'. ".

Finally, before leaving this summary of Dr. Johnston's first two articles, there is one point that might require clarification for the reader who may not have extensive background or training in these matters. Dr. Johnston uses the terms "enterprise key" and "surrogate key" in a way that may lead the inexperienced reader to conclude that somehow an enterprise key is not a surrogate key. A surrogate key is an unintelligent key- meaning a key whose value is devoid of any business meaning. An enterprise key as presented in the context of this paper is also designed to be devoid of any business meaning and is, therefore, also a surrogate key. The distinction that Dr. Johnston is making when he uses these terms relates to the level at which these keys enforce uniqueness in their domain of values. He uses surrogate key as the term for describing a primary key whose domain of valid values is guaranteed to be unique only at the entity or table level of a database. An enterprise key is a surrogate key whose values are guaranteed to be unique over all the entities or tables that comprise the entire enterprise environment of distributed databases. However, this is not the only distinction that Dr. Johnston makes between a "plain vanilla" surrogate key and an enterprise surrogate key.

Dr. Johnston's concept of an enterprise surrogate key as a primary key requires that it possess two additional properties that are not required of a primary key according to relational theory. Recall from Appendix D, in the discussion of the properties of a relational primary key that the three mandatory properties that a primary key must possess are that it be unique, unchanging, and mandatory (nulls not allowed). [Other properties such as unintelligent (no business meaning), minimalness (fewest necessary attributes), and instant availability may be desired but are not mandatory.] Under this set of mandatory constraints, surrogate keys composed of multiple attributes (columns of tables) are permitted allowing for the possibility of surrogate primary keys from different tables to have differing structures (syntax). In addition, the uniqueness property required by relational theory is only enforced at the entity (table) level. The two additional mandatory properties of an enterprise surrogate key defined by Dr. Johnston require that, first, the structure of all enterprise keys be common, composed of a single attribute of the same data type and length, and second, that the value of every instance of an enterprise key be unique across every row of every table in the enterprise information system environment.

The subject matter of Dr. Johnston's four remaining articles titled, *De-Embedding Foreign Keys*, addresses the problem of how to deal with the ripple effect of existing business primary keys that ripple through the tables of databases as the implementers of the relationships between those tables. This effect is generally the issue that is at the core of the concerns that relational data modelers usually express about the idea of reengineering database schemas to implement enterprise keys. The relationships expressed by these embedded foreign keys represent the business rules of the enterprise. Any reengineering of the structure of the schema must preserve these rules, otherwise the schema will not be an accurate reflection of the true data requirements of the enterprise. Dr. Johnston develops two technically feasible methods for dealing with the issue.

The presentation of these methods is somewhat more technical in nature than the previous two articles on the problem and the solution, and will only be summarized here. The first describes how foreign keys can be de-embedded and at the same time preserve the business rule relationships through the creation of metadata tables to keep track of the tables in the database and the relationships between them. The second method acknowledges the limitations of current standard SQL (the lingua franca of RDBMSs) for supporting a de-embedded database and proposes a technique for overcoming the limitation by creating a mirror database of a de-embedded database as the interface to the user. The mirror database would be semantically equivalent to the de-embedded database but physically different as the foreign keys would be re-embedded so that existing SQL queries would function correctly. This method is feasible but also more labor intensive and costly as a consequence of the effort that will be required to maintain the mappings between the two databases whenever a change occurs to a relationship in the de-embedded database.

H.3 Surrogate Key References

In contrast to the dearth of references on reengineering with enterprise keys, there is voluminous repetitive materiel available that deal with the general subject of surrogate keys. Most references typically begin by defining a surrogate key as an unintelligent key containing no business meaning and then listing the advantages of surrogate keys and contrasting them with the advantages of intelligent business keys. Advantages and disadvantages typically cited are as follows[75]:

- **Advantages of Surrogate Keys**
  - Can easily be system generated without user involvement.
  - Immediately available at create/insert time.
  - Stable values not dependent on business rules or usage.
  - Compact usually composed of integers requiring minimal storage.
  - Uniform data type facilitating generalized reference structures.

- **Disadvantages of Surrogate Keys**
  - Obscure values with no meaning in a business context.
  - Require more code intensive SQL queries.

75 "Technical Keys", http//:www.aisntl.com/case/technical_keys.html.

- Join intensive queries can adversely affect database performance.

In the early days of the development of relational database management systems, there was no firmly held consensus among relational data modelers as to whether unintelligent surrogate keys or intelligent business keys were to be preferred for primary keys. Opinions tended to be about evenly divided between advocates for surrogate keys and advocates for business keys. Those practitioners favoring business keys generally cited the DDL/DML coding complexity and database performance penalties attributed to the use of surrogate keys, while extolling the user friendliness of natural or business keys. Proponents of surrogate keys praised their stability and independence from changing business factors and the compact, uniform structure of surrogate keys in contrast to the instability of business keys. However, in recent years the weight of opinion among relational data modelers and database designers has swung in favor of the use of surrogate keys for the reasons cited in the previous section. Namely, that the growth of large scale information systems, in a dynamically changing business environment, has exposed the vulnerability of intelligent business primary keys to changes in their values, thus violating one of the mandatory properties of a primary key that values may not change once assigned. The consequence of this is compromised referential integrity threatening the reliability and accuracy of business data and information.

In addition to Dr. Tom Johnston and Mike Lonigro, many other well-known professional database designers such as Dr. Ralph Kimball and Terry Moriarty favor the use of surrogate primary keys for relational databases. Dr. Kimball is a nationally recognized expert database designer with over 30 years of professional experience in the industry. He was the co-inventor of the Xerox Star Workstation, the founder and former CEO of Red Brick Systems, inventor of the Star Schema relational database design, author of numerous books on designing databases including *How to Design Dimensional Data Warehouses* and *The Data Warehouse Toolkit*, and a strong proponent of surrogate keys. In his Data Warehouse Architect Column in the May 1998 on-line *Database Programming and Design*[76] Dr. Kimball states:

> *"Actually, a surrogate key in a data warehouse is more than just a substitute for a natural key. In a data warehouse, a surrogate key is a necessary generalization of the natural production key and is one of the basic elements of data warehouse design. Let's be very clear: Every join between dimension tables and fact tables in a data warehouse should be based on surrogate keys, not natural keys. …the keys are surrogate keys that are just anonymous integers. Each one of these keys should be a simple integer starting with one and going up to the highest number needed."*

---

76 "Surrogate Keys", Ralph Kimball, *DBMS On-Line,* May 1998 at http//:www.dbmsmag.com/9805d05.html [Used by permission of Dr. Kimball].

Dr. Kimball makes a strong case for the use of surrogate keys based on the instability and variability of business keys which he calls "production" keys. This is what he says:

> *As the data warehouse manager, you need to keep your keys independent from the production keys. Production has different priorities from you. Production keys such as product keys or customer keys are generated, formatted, updated, deleted, recycled, and reused according to the dictates of production. If you use production keys as your keys, you will be jerked around by changes that can be, at the very least, annoying, and at the worst, disastrous."*

Terry Moriarty is also a nationally prominent data modeler and database designer. She is president of Inastrol, an information management consulting company, and is also a frequent lecturer on data management topics at industry conferences and forums. In addition, she is the author of a column in *Intelligent Enterprise*, a magazine for professional data administrators. Ms. Moriarty was contacted through the Data Management Discussion Group seeking advice on where to find sources for reengineering with enterprise keys. This is what she had to say in her e-mail reply concerning surrogate keys[77]:

> *"I recently worked with Mike Lonigro. He does believe in the use of surrogate keys, as do I. …*
>
> *I believe in using surrogate keys exclusively. Using natural business data keys can really get you in trouble, as the business changes. First, few 'natural identifiers' are really unique. There is almost [always] some set to which the identifier doesn't apply. Social Security Number is a perfect example. The belief is that every person must have one, but that is not true. Non-resident aliens do not have to have an SSN, unless they work in the US. Also SSNs are not unique to a business party, the same one can be used for both a person and the unincorporated business that person owns.*
>
> *I also find that the business often puts meaning into the identifiers that they assign. For example, customer IDs may include a digit that indicates what market the party belongs to. The market changes, the customer ID changes.*
>
> ***I just think we are safer to keep business identifiers as just data about the business, enforce the appropriate uniqueness business rules on those***

77 E-mail Index# 1259, Data Management Discussion Group, http//:www.egroups.com/group/dm-discuss/1259.html [Used by permission of Ms Moriarty].

identifiers and use surrogate keys internally for referential integrity*".*
[Boldfaced emphasis added]

Ms. Moriarty's last sentence is emphasized because it is a very concise and simple expression of exactly what enterprise keys as proposed in this concept paper are designed to accomplish for Army Logistics information systems.

APPENDIX I

SUMMARY OF PREVIOUS INITIATIVES FOR IMPLEMENTING ENTERPRISE IDENTIFIERS

APPENDIX I – SUMMARY OF PREVIOUS INITIATIVES FOR IMPLEMENTING ENTERPRISE IDENTIFIERS

Background

The ability to identify records in a globally unique manner is an issue of paramount importance when information resides in multiple databases that need to exchange data and maintain referential integrity when updating their tables. It is also an essential component to the future of information systems which quite likely will form part of virtual distributed databases operating in web-based, web-like environments.

The Army recognized this requirement early and under the auspices of the Military Communications Electronics Board (MCEB) convened a Joint Information Process Team (IPT) to develop an implementation plan for organization identifiers or ORGIDs (See "Organization Identifier Implementation Plan" in paragraph I.2 below). Afterwards, the Army submitted a proposal to include in the DoD data standard, Organization Identifier the data type "Bit-String" to complement the existing type "Character-String" (See paragraph I.3, "Point Paper, December 3, 1998"). This change was subsequently approved. As a result, the DoD standard data element ORGANIZATION IDENTIFIER permits the use of a 32 bit integer for the purpose of uniquely and globally referring to all pertinent military units. Finally, the Army Research Lab developed a prototype ORGID server that can provide to any authorized requestor, on a first-come-first-serve basis, a sequentially generated 32 bit key to be used in the creation of force structure tables (See paragraph I.4, "Organization ID Server to Organization Server Interface").

A prototype is required to demonstrate the feasibility of using this 32 bit integer key to create Enterprise Identifiers made up of the 32 bit unique key of the organization and a concatenated second 32 bit integer generated locally by the authorized organization for the purpose of uniquely identifying data created under its control. In other words, the prototype will demonstrate that globally unique identifiers can readily and in an orderly fashion be constructed by those organizations that own the data needed for exchange so that all participants in data exchange with this organization can query any record so identified with the assurance that (1) the key structure will always be the same in all the databases that collect and maintain that information and (2) updates to the values of the record can be effectuated in an efficient and reliable manner.

The following sections document some of the proposals for use and implementation of key management schemes akin to those described in the main portion of this document.

I.1 Proposal By ODISC4 to Army Architects For Key Management

In recent months, the Army architects involved in the development of CADM compliant databases for the purpose of developing all future architectures for the Army considered different alternatives to key management. At a minimum they have agreed that the key management must be in the hands of those who own the data, and to that effect have identified "ownership" of each of the tables that will be populated in these CADM compliant databases, and which will be involved in data exchange among the participating organizations.

Because adoption of a common approach to the key management problem is inherently more efficient for implementation, PEO C3S has proposed that all records in the pertinent tables be given identifiers conformant with the same generating scheme.

I.1.1 Comments

However, the scheme is at variance with what the prototype funded by the Army, and tested by USAFMSA for the purpose of populating their force structure tables, has adopted. In particular, the PEO C3S does not take advantage of the DoD standards already in place for how to identify military units via ORGANIZATION IDENTIFIER, since it uses a "12 digit" to point to all the military units instead of a 32 bit integer. It also does not take advantage of the ORG-ID server that has already been prototyped, and which could readily provide the required identifiers for all the military organizations that the Army architects plan to capture in their future architectures. Furthermore, by wedding itself to a digit oriented representation of the identifiers, as opposed to a bit oriented one, it may impose greater bandwidth requirements if architecture information must be passed using communication lines with reduced capacity, such as those which for the foreseeable future will be part of the Army, and DoD infrastructure. And, last but not least, it creates yet another set of identifiers for what already other organizations such as USAFMSA plan to use in identifying an essential portion of the battlefield resources, namely, the military units that commanders need to manage when prosecuting the battle and forming units on the fly.

I.1.2 Recommendations

Given that (1) a scheme for identifying organizations has already been demonstrated, and is quite likely to become the way in which USAFMSA will manage all military units down to the billet level, it appears that the creation of yet another set of identifiers that will not take advantage of this approach is duplicative and likely to make data exchange among all Army systems more complex than it need to be. (2) The PEO C3S scheme is also at variance with the way in which the DoD standards recommend that ORGANIZATION IDENTIFIERs be constructed.

For these reasons, it is recommended that:

- Applicability of the Enterprise Identifier scheme, as it has been demonstrated in the ORG-ID prototype, be considered as an alternative to the current proposal[78].

[78] A 32-bit integer could be displayed as a hexadecimal string. Thus, an organization whose identifier is the bit integer corresponding to 2^32-1 (or 4,294,967,295 in decimal notation) would be represented as FFFFFFFF. The 32 bit integer scheme that the Army has already prototyped allows for the generation of a total of 4,294,967,296 unique keys. Organization A whose ORGANIZATION IDENTIFIER happens to be FFFFFFFF could then concatenate another 32 bit integer, sequentially generated, to the value of its unique ORGANIZATION IDENTIFIER and manage up to 4,294,967,296 records, starting with FFFFFFFF00000000 all the way to FFFFFFFFFFFFFFFF. These keys are (a) one half the length of a 32 character string or a 32 integer representation, (b) occupy half the memory space in the computer, and (c) can be more easily transferred over narrow bandwidth communication lines, than a key having a structure such as "345664439492-

- At a minimum, adoption of bit oriented, as opposed to integer/character oriented identifiers, for architecture data be also considered to ensure that data exchanged will not place an excessive burden on the communication infrastructure that the Army will likely use in the foreseeable future.

- Implications of implementing yet another way to identify organizations within the Army be taken into consideration.

I.1.3 Addendum

The wording of the recommendations may have given rise to a false impression, namely, that there is a need for a "foreign key" or an "extra key" in the current structures. This impression may have been caused by the reference to ORGID in the write-up. Note that ORGID is only mentioned to clarify "how" the keys are to be constructed.

The basic idea is as follows:

(1) Some organization A, that has been given ownership of a particular table(s) in the Army CADM, agrees to implement the ODISC4 scheme. To do so it goes to the force development community (e.g., USAFMSA) and requests that an organization entry be created in the Organization server that will include a globally unique 32 bit integer to unique identify it. USAFMSA creates the organization and sends the ORG-ID to organization A.

(2) Upon receipt of the 32 bit ORG-ID that is owned only by Organization A (i.e., no other organization in the Army will have a 32 bit integer identical to the one of Organization A), Organization A proceeds to generate 64 bit integer keys for all the records that it creates (i.e., has assigned ownership) by taking the 32 bit integer that it received from USAFMSA and concatenating it with another 32 bit integer. The keys for the records will all have the same structure i.e., a 32 bit half that is numerically identical to the 32 bit value that USAFMSA gave to Organization A as its identifier, plus another half which is a sequential number starting at 0 and ending at 4.3 billion. Note, therefore, that it is just an artifact of the key generation scheme that the first half will be numerically identical to the 32 bit integer that USAFMSA uses to point to that organization. The key generation scheme is not, nor is intended to be, a foreign key from ORGANIZATION to the particular table that Organization A owns.

(3) Every other organization that also agrees to implement the scheme will then go through the steps that Organization A followed and in turn will be able to manage, and, with ontological certitude, generate keys for up to 4.3 billion records that will not collide with any key from the other tables. This is because the first half of the 64 bit integer is already guaranteed by USAFMSA to be unique throughout all the participating organizations, so if the second half

---

39390002039939993929" where the first 12 digits point to the ORGANIZATION and the second is generated by some other means that ensures uniqueness, such as invoking the built-in function `GETDATE()`. Generation of Hexadecimal Representations (HR) is straightforward and most applications such as MS Excel have built-in functions that take a decimal number and give back its HR equivalent. This could be used to readily populate any table that adopts this scheme, or to generate more user-friendly, human readable outputs.

is a sequential number that is never repeated, then it too must be universally unique. This occurs every time a new record is created in a table that Organization A owns.

So, for example, if Organization A owns Table 1 and Table 2 of the Army CADM, then it can label every record in its two tables in a globally unique manner by implementing the scheme mentioned above, e.g.:

CADM TABLE 1 (owned by Org A with ID=FFFAB001 it received from USAFMSA)

EID	Table1.Attribute 1
FFFAB00100000001	XYZ1
FFFAB00100000002	FRD3
FFFAB00100000006	ADC8

CADM TABLE 2 (owned by Org A with ID=FFFAB001 it received from USAFMSA)

EID	Table2.Attribute 1
FFFAB00100000003	XPZ1
FFFAB00100000004	ARD3
FFFAB00100000005	MDX8

And if Organization B has ownership of Table 11 of the CADM it can label its records using the same approach and no record so labeled will ever conflict with the key for another record in any of the participating databases:

CADM TABLE 11 (owned by Org B with ID=FDFAB001 it received from USAFMSA)

EID	Table11.Attribute 1
FDFAB00100000001	XT11
FDFAB00100000002	PRD3
FDFAB00100000003	AAA8

I.2 Excerpt from the Organization Identifier Implementation Plan Dated August 27, 1998

I.2.1. Purpose

This plan describes how to define, develop, implement and maintain a standard Organizational Identifier (ORGID) system. The purpose of the ORGID is to provide an operationally useful standard for identifying an organization for machine-to-machine data exchange and improve interoperability, minimize communications loads for tactical mobile systems, and improve automated system processing and data exchange.

I.2.2. Objective

Currently, the ORGID in the Defense Data Dictionary System (DDDS) is defined as a 15 character field with no meaningful values. It was defined at the conceptual level and is not generally operationally useful. The objective is to standardize the ORGID as a 32 bit surrogate key for use in system databases.

The recommendation is to use a surrogate key as the primary key for organizations. In this case, a four-byte (32 bit) unsigned integer. A 32-bit unsigned integer has the capability to uniquely enumerate more than 4.3 billion (actually, 2^{32}) organizations. This approach has four obvious advantages: (1) it is very general; (2) it provides a very terse manner in which to exchange organization identities, (3) it prevents waste of ORGID values, and (4) it allows the computers to do simple integer operations that are very fast, because all references to organizations are handled as integers. For example, a complete tree structure can be built by simply including an integer attribute in each organization entity that references that organization's parent organization (be it the default, current, or some other definition of parent). This allows very quick access up and down the organization tree using breadth-first or depth-first searches.

Within the Variable Message Format (VMF) Technical Interface Design Plan (Test Edition)(TIDP-TE), Reissue 3 dated 18 May 1998, there is a Data Use Identifier (DUI) titled Unit Reference Number (URN). It is 24 bits in length allowing over 16 million unit unique identification codes. This is a reference number used by units in a VMF interface to uniquely identify friendly military units. The URN concept was developed by the service/CINC representatives of the VMF Subgroup to serve as a unit identifier for all VMF messages where a unit organization is required. In addition, the same URN was proposed and accepted as a unit identifier for MIL STD 2045-47001, an application layer standard for communications over Tactical Internet. The ORG-ID proposal builds upon the URN by making it a subset of the ORG-ID, thus taking advantage of the URN efforts. URN is 24 bits allowing up to 16 million unit unique identification codes. The URN was initially developed as part of Army digitization in support of battlefield situational awareness. In 1995 the Army proposed to the Joint Variable Message Format (JVMF) sub-working group that URN should be established as a unit identifier for all variable message formats. In addition, the same URN was proposed and accepted as a unit identifier for MIL STD 2045-47001, an application layer standard for communications over

Tactical Internet. The intent is for every organization to include units to be assigned an ORG-ID with the overall result of:

a. Providing command elements with detailed situational awareness
b. Reducing the possibility of fratricide
c. Providing greater interoperability between all systems
d. Minimizing the cost of interoperability between systems

I.2.3. Implementation

The new ORGID will consist of the URN plus 8 future-use bits. URN is used as an identifier for all message-based systems that are bound by communications limitations, while ORGID is used for database to database transfer over high speed communication protocols. The CINCs, Services and Agencies have provided tables assigning URNs to the units for which they are responsible. This list will be centrally maintained and will be Appendix B, Part 2 of the VMF TIDP-TE.

URNs have been allocated as follows: 2 million URNs for Army use, 1 million each for the Navy, Air Force and Marines, 10,000 for the CINCs and 990,000 CINC/JTF spares. Each service was allocated more URNs than they can currently consume. ORG-ID users will use URNs that have already been assigned. Additionally, over 10 million URNs were reserved for future growth, including allocation to coalition forces. However, if ever the current URNs are completely assigned, the intent is to expand unit identification using the eight additional bits under ORGID. Until that time users of the URN will continue to use 24 bits.

Every entity in a database must have a unique (primary) key to identify it; this includes organizations. Often, to create uniqueness, primary keys are created by combining primary keys from other entities. This can be illustrated by using the example of software versions. The name of software is unique until additional versions are created for the same name. For example, SoftwareA, then SoftwareA, V1.0; SoftwareA, V2.0 and so on. To be unique, both the name and the version represent the key. Although the term "naming convention" typically brings to mind human readable forms of creating and identifying unique organization names, this is not necessary for computers. From a data- processing perspective, there is nothing simpler than an integer. Surrogate keys are primary keys that have no business meaning; a good example of this is an integer. Integers offer several advantages, including simplicity, fast performance, and easily verifiable uniqueness. Normally, users will never know surrogate keys exist, but the database management system and application programs can use them extensively to greatly enhance performance and flexibility.

Since humans do not always relate well to numbers, there can always be alternate (primary) keys stored as attributes to the organization entity. For example, an organization name. However, a basic assumption is that in the future, operators will not be accessing databases directly but, rather, will be using application programs that in turn access databases. Thus, operators will be insulated from details like surrogate keys.

Administration of this approach is done with a two layered set of servers: ORGID servers and organization servers. The purpose of ORGID servers is to ensure that ORGIDs are unique and not wasted. The plan is for the ORGID servers to be controlled by the DOD-CIO. Actual organization information would be maintained in organization servers controlled by the individual organization proponents (e.g., services, other government organization, non-government organizations, other countries, etc.). To find detailed information about organizations, one would contact one of the organization servers.

When an authorized Organization server wants to create new organizations, ORG-IDs must be obtained from an authorized ORG-ID server. This is accomplished by sending a request to an ORGID server which would (1) return a set of ORGIDs to the requesting organization server, and (2) track to whom the ORGIDs were given and mark those ORGIDs as used but inactive. Within some maximum time, the Organization server must activate the ORGIDs by notifying the ORGID server that the organizations associated with the ORGIDs have been created. Disestablishment of the organizations works in a similar matter. The goal is to eventually recycle unused ORGIDs.

The actual organization information is located in “organization servers” maintained by the individual services or agencies. Each organization logical server database is modeled after the Command and Control Core Data Model (C2CDM) organization view. The organization servers contain the detailed description of the organization. For example, the ORGID server would keep track of the fact that ORGID 123456789 is an active ORGID that belongs to the “army.mil.us” domain. To find detailed information about organization 123456789, one would contact one of the army.mil.us organization servers.

More information could be stored at the ORGID servers. For example, to maintain an authoritative tree structure of the current known organizations, two other attributes could be maintained: (1) a character string for the name of the organization and (2) the ORGID of the organization’s administrative parent organization. This would allow a skeleton tree structure to be built that could assist with validation and error checking. The reason for not proposing this is that a country or service may not want to share its complete organization tree. Separating the assignment of ORGIDs from the information about the organization to which the ORGID has been assigned allows one to selectively share information about its organizations while still participating in the open, ORGID assignment system (an alternative approach is to block allocate ORGIDs, and that will cause waste). It is expected/hoped that the separation of ORGID assignment process from the maintenance of organization information will encourage participation in the use of the ORGID servers by those who would not otherwise do so due to sovereignty or security reasons.

The issues surrounding ORGID stability have significant implementation ramifications. At this point, there may be a vision of a huge bottleneck being created as thousands of users flood an ORGID server as they attempt to create new organizations during a battle. A basic assertion is that ORGIDs are very stable because they are assigned based upon a stable administrative (or default) chain of command. Dynamic, operational chains of command are built through the re-linking of existing, administrative organizations. This means that in nearly all cases, building a new task organization does not require the generation of new ORGIDs. Only special cases, like the creation of a Joint Task Force (JTF), may require an ad-hoc visit to the ORGID server (even this can be circumvented by pre-allocating a few ORG-IDs to the unified commands). Therefore, creating the stable set of “functionally static” ORGIDs is primarily an administrative function, akin to the building of the Table of Organization and Equipment (TOE) in the Army, Unit Manning Document (UMD) in the Air Force, and of Ship Manning Document (SMD) in the Navy.

In the vast majority of cases, the building of a new task organization simply requires that existing organizations, with stable ORGIDs, be temporarily associated with new parent organizations. In other words, new operational structures are built with existing organizations. In an Army example, two battalions may switch companies to build a battalion task force with a temporary name of “Task Force Alpha.” In a Navy example, a specific destroyer squadron, two submarines, and a supply ship are temporarily attached to a core carrier battle group. (Recall that a ship’s crew is represented by an umbrella organization.) In a Marine example, a battalion landing team is constructed and temporarily attached to a Marine Expeditionary Unit (MEU). In an Air Force example, a strike package is created by combining together organizations that each represent an aircraft and its crew. In these

The assertion in the previous examples is that none of the task organizations examples required the creation of a new organization but, rather, only the restructuring of the links to existing organizations. Although the associations are dynamic, the existence of the individual organizations is static. The details of Air Force strike package example are still being debated and currently represent the most difficult challenge to the stability assertion of ORGIDs.

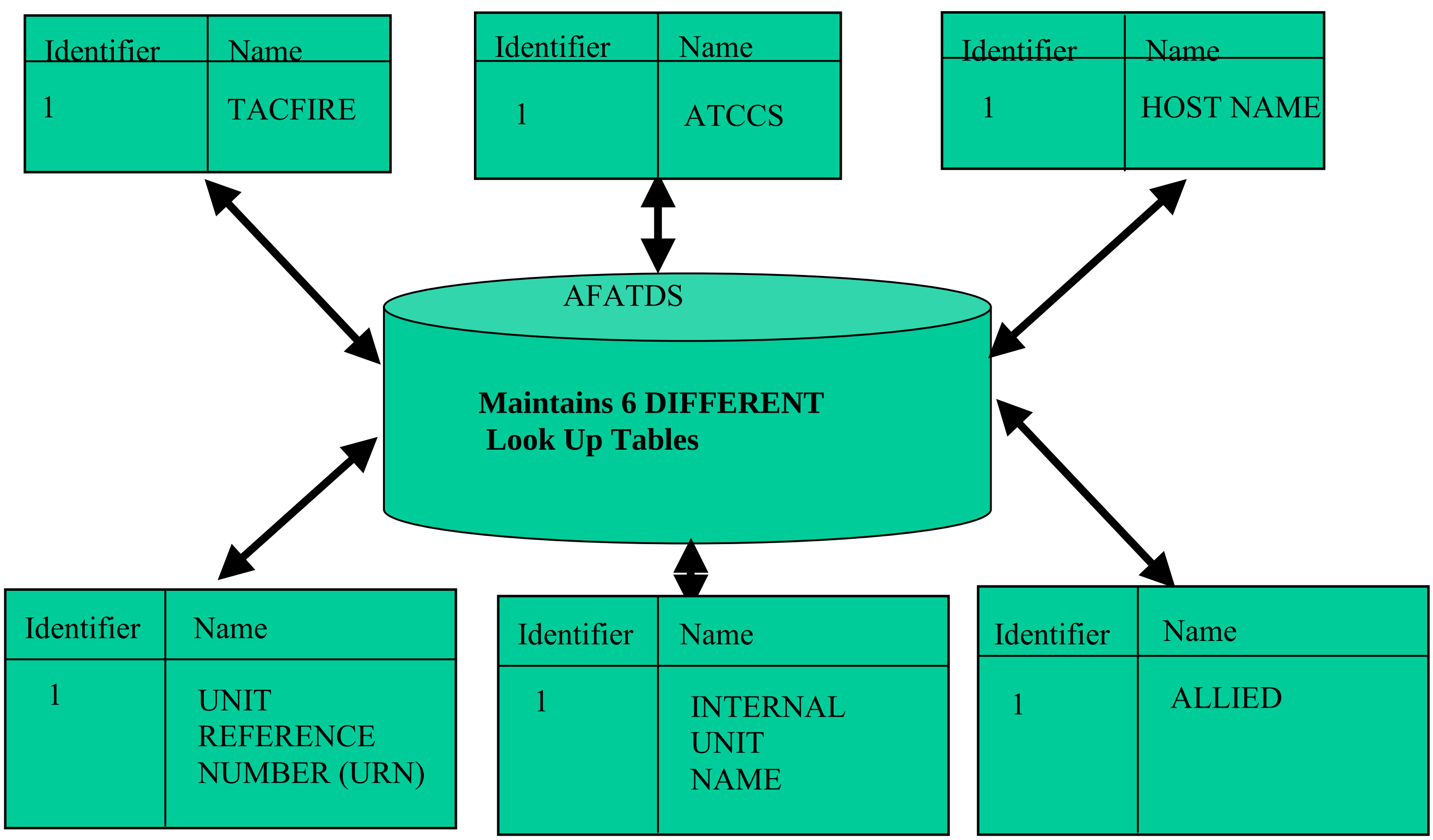
I.3 Unique Organizational Identification Point Paper

I.3.1 Purpose

To change the data type of ORGANIZATION IDENTIFIER from Character-String to Bit-String.

I.3.2 Facts

The combat commander needs information about the units (organizations) around him – friendly, enemy and unknown, as well as things above and below him (e.g., airplanes, helicopters, submarines). To do this requires that each battlefield resource be uniquely identified so that information about them can be queried and displayed. Currently, C4 systems (e.g. AFATDS, ASAS, MCS) must store and update half a dozen or more different homegrown tables to identify Units (see below). This is analogous to each U.S. State maintaining its own Zip code structure! For example, the Army Field Artillery Tactical Data System (AFATDS) must use organization data from six different sources (i.e., TACFIRE, ATCCS, Unit Reference Number, Allied). Each represents organizations in a different way. AFTADS must convert them before use. Clearly, if Army and DoD interoperability goals are to be achieved, a universal, unique identification scheme **must** be established.



For almost two years the Army has been trying to make ORGANIZATION IDENTIFIER, a DoD Standard Data Element, operationally useful. ORGANIZATION IDENTIFIER is currently defined as "The identifier that represents an administrative structure with a mission." It is a 15 character field with a domain definition of the ASCII graphic character set. It has no homogeneous set of unique domain values that are acceptable DoD-wide. If it does not have any universally acceptable domain values, how can the Army or anyone else use it?

ORGANIZATION IDENTIFIER and other identifiers were created by the DoD functional community and approved by the DoD Data Administrator at a considerable cost, but they have little or no use in information systems. There are 1616 other approved Standard Data Elements using IDENTIFIER in the Defense Data Dictionary System (DDDS). Although many of these have "real world" domains and are implemented in physical databases, there are a number that are only conceptual and lack any implementation plan (e.g., PLAN IDENTIFIER (5644), TASK IDENTIFIER (9282), PERSON IDENTIFIER (11185), DOCUMENT IDENTIFIER (9643), ACTION IDENTIFIER (9904), SITUATION IDENTIFIER (9909), ACTION-OBJECTIVE IDENTIFIER (11173), FACILITY IDENTIFIER (11179), FEATURE IDENTIFIER (11180), MATERIEL IDENTIFIER (11182), CAPABILITY IDENTIFIER (11287), and so on).

Looking at the above examples, you can see that these identifiers do not represent trivial objects of interest. In a database they are designed to identify a unique occurrence of a plan, task, person, etc. However, when several roles relate to the same object and information about similar objects is shared across functional areas, overlapping meanings cause problems in identifying object uniqueness. Another example, social security number (SSN), is a unique way to identify a US citizen (PERSON), but then so is patient number in a closed hospital environment. The intent here is to show that many of these identifiers (e.g., ORGANIZATION IDENTIFIER, UNIT IDENTIFIER) exist in the DDDS with no domain they can call their own, because they co-exist with other identifiers whose domains are redundant or partially overlap with theirs. There is no one standard business-based domain for each of them.

The Standard Data Element ORGANIZATION IDENTIFIER (ORG-ID) is constructed using the Class Word/Generic Element IDENTIFIER. DoD 8320-1-M-1, Data Standardization Procedures, defines CLASS WORDS and relates them to GENERIC ELEMENTS as follows: "A class word is a noun that designates the general category of data at the highest level and subcategorizes data elements based on like meta-data [information about data]. Class words, with or without modifiers, are known as generic elements…" (AP5.1.4.1., p. 91).

A Generic Element in the DoD Data Administration program defines the structure and type of the data value domain for a data element. DoD 8320.1-M-1 divides generic elements into two classes: qualitative elements and quantitative elements. The data types Character-String and Bit-String are available in the DDDS for all qualitative elements (NAME, TEXT, CODE, IDENTIFIER). Because ORGANIZATION IDENTIFIER is a qualitative data element, it may change its data type from Character-String to Bit-String. The data type of a data element using the generic element IDENTIFIER can be changed from Character-String to Bit-String at the data element level. The Character-String reflected at the generic element level is only a default and can be changed to fit the requirement of the data element. The same is true for the domain definition text. What appears at the generic level is a default where the domain definition at the data element level can be changed to reflect the requirement. In other words, ORGANIZATION IDENTIFIER can have the domain definition: "Four bytes unsigned integer (32 bits)."

The domain definition for IDENTIFIER is "A combination of one or more numbers, letters or special characters which designates a special object/entity, but has no readily definable meaning" (AP5.5., p. 95). Clearly, ORGANIZATION IDENTIFIER inherits its meaninglessness from IDENTIFIER. IDENTIFIERs do not have specific domains. All identifiers in the DDDS have general domains, and they contain no listing of domain values. The comment text for IDENTIFIER states "This Generic Element cannot be used for mathematical computations. A standard data element that uses this Generic Element to establish its class of data must have a GENERAL domain (includes designator)."

I.3.3 Key Points

I.3.3.1 Problem

There are many methods for identifying organizations and units. There is no single representation of the concept of organization and no universally accepted unique identifier for it. The scope of other identifiers, such as UNIT IDENTIFIER, is insufficient. The lack of a unique ORG-ID has been the cause of many identification problems within the DOD. Solving this problem is a requirement of the Army, but JCS recognizes it as a Joint problem. A solution must be acceptable to all services as well as DOD agencies.

I.3.3.2 Data Administration Approach

The language in the DoD 8320.1-M-1 gives a "fix" to the IDENTIFIER problem. It states in paragraph C5.2.3.5. (p. 38): "Any data elements using the class word "IDENTIFIER" and proposed as primary key attributes must represent "real world" identifiers and be unique across the DoD. The Authority Reference Text, cited for these IDENTIFIER data elements and documented in the DoD data dictionary, should contain the justification for the use of the identifier and the method for how it is created and maintained. If the Authority Reference Text does not provide this information, the method and/or plan for creating and maintaining the identifier should be documented in the DoD data dictionary in the data element Comment Text. (See AP1. Appendix 1 for the definition of the data element meta-data requirements, Authority Reference Text, and Comment Text.)."

1) For many of the DDDS identifiers, as already stated, there is no one unique "real world" domain; instead, there are multiple overlapping domains. A solution to the IDENTIFIER problem, from an information system/database perspective, is the use of a surrogate key. A surrogate key is a unique way of identifying an instance or occurrence of the data, but the surrogate key has no business meaning. It represents one or more "real world" identifiers; it is not a "real world" identifier. Commonly, a surrogate key is a sequential number that is attached to what is being identified. For example, there is a heterogeneous group of people: US new born children, US citizens, and foreign nationals. The requirement is to store them all in the same database table. There is no unique piece of data that will identify each of them. To make this work, a surrogate key is attached to each person's record. Some persons might have a unique means of identification such as SSN; this becomes an alternate key (An alternate key is a candidate for primary key, but not selected as the primary key):

2345	John Smith	222458567	(US citizen)
2346	Shu Chen		(Foreign national)
2347	Baby Jones		(US new born, no SSN yet)
2348	Hans Reubens		(Foreign national)
2349	Bill Jones	078324517	(US citizen)

Note: 2345, 2346, etc. are instances of the surrogate key.

No matter who these people are or what status they may have, the surrogate key, transparent to them and the users of the data, is a unique identifier.

2) The surrogate key solution for ORGANIZATION finds support both within and outside DoD. For example, twenty-two of the existing IDENTIFIER Standard Data Elements in the DDDS contain comment text stating that they are surrogate keys (e.g., PERSON-TYPE-MATERIAL HOLDING ESTIMATE IDENTIFIER (11974), PERSON-TYPE-ORGANIZATION-HOLDING-ESTIMATE IDENTIFIER (11982), PERSON-TYPE-FACILITY-HOLDING-ESTIMATE IDENTIFIER (11961), etc. Recently, Ms. Judith Newton, editor of the International Standards Organization (ISO) Standard on Specification and Standardization of Data Elements: Naming and Identification Principles for Data Elements (ISO/IEC 11179-5), wrote: "Since many names may be associated with a single data element, it is important to also use a unique identifier, usually in the form of a number, to distinguish each data element from any other. ISO 11179-5 discusses assigning this identifier at the International registry level…"

3) The easiest to implement and most machine-efficient surrogate key is the (binary) integer. Integer is represented in bytes. One byte is 8 bits. Being at the computer's lowest level (bit) of processing creates the efficiency. As mentioned earlier, in the DDDS the Generic Element for IDENTIFIER allows both Character-String (ASCII) and Bit-String (Integer). Changing the data type of the data element ORGANIZATION IDENTIFIER from Character-String to Bit-String does not affect other identifier data elements using the character-string data type. Such data elements can still have domains using alphabetic, numeric or special characters.

4) The proposal is to change ORGANIZATION IDENTIFIER from a Character-String of length 15 to a Bit-String of length 32 serving as a surrogate key (no business meaning). A 32-bit unsigned integer has the capability to uniquely enumerate more than 4.3 billion organizations (actually, 2**32), provided that all the possible numbers are used.

5) Integer values are assigned first-come, first-served, with no waste. This has three obvious advantages: (a) it is very general; (b) it provides a very terse manner in which to identify organizations, and (c) it allows the computers to do simple integer non-computational operations that are very fast.

6) There can be alternate (primary) keys stored as attributes to the organization entity. A basic assumption is that operators will not be accessing databases directly but will use application programs that in turn access databases. Thus, operators will be insulated from details like "surrogate keys."

7) Administration of this approach is done with a set of seamless ORGID servers. The ORGID servers are developed and maintained by the DOD CIO. When an authorized user wants to create a new organization, an ORGID must be obtained. A request is sent to one of several ORGID servers. The requestor must return to the server the status, active or inactive, of the ORGID requested. The purpose of an ORG-ID server is to ensure that ORGIDs are unique. The actual organization information is located in "organization servers" maintained by the individual services or agencies.

I.3.4 Conclusion

In conclusion, we recommend that the Functional Data Administrator (FDAd) for ORGANIZATION IDENTIFIER approve version 2 (counter 7875), which changes the data type for ORG-ID from Character-String to Bit-String.

I.4 Organization ID Server (OIS) to Organization Server (OS) Interface[79]

I.4.1 The OIS to OS Interface

The OIS to OS interface has several parts. At the bottom, handling the details of the network connections is a software package called PKG. This software was developed by personnel from the Army Research Laboratory and implements a message passing interface for managing client/server communications using TCP/IP connections. Server actions include initialization, client connection, and client request servicing. Client actions include making the server request, reply processing, possible connection negotiation, and data transfer processing. Above this are a set of OIS routines that prompt and process PKG input and output and finally, there are a library of functions that comprise the API and are called directly by the OS program.

I.4.2 The API Library Functions

The API library of functions supplied to the Organization Server developer will handle the details of the communication with the OIS and provide a set of functions allowing the OS to interact with the OIS. At present, the following actions are allowed:

(1) The **`ois_login`** function creates the connection to the OIS and attempts to login using the approved user name (uname), and password. Return values indicate the status of the connection (PKG_FAIL, LOGIN_FAIL, LOGIN_COMPLETE).

```
int ois_login ( host_p, host_s, uname, password, debug, eflag )
   char *host_p;   /* Name of ORG-ID SERVER primary host machine */
   char *host_s;   /* Name of ORG-ID SERVER secondary host machine */
   char  *uname;         /* Name of OS or other approved user */
   char *password; /* Approved passwd */
   FILE *debug;          /* File descriptor for debug information.
                            May be NULL */
   int eflag;      /* Flag for encryption. 1 is on 0 is off */
   returns:
        BAD_ARGUMENT – One of the input arguments didn't make sense.
        Generally this is a pointer being NULL or a variable being
        out of reasonable range.
        PKG_SEND_FAILURE – PKG software send failed for the specified
        connection.
        PKG_WAIT_FAILURE – PKG software failed to get an expected
        response on the open connection.
        LOGIN_OK_P – Login to the primary server was successful
        LOGIN_OK_S – Login to the primary server failed but a
        successful login to the secondary server was completed
        PKG_CONNECT_FAIL – PKG software failed to connect to the
        specified host
```

[79] Excerpted from ARL Technical Report: "A Primary Server for Organizational Identifiers," by Fred Brundick and George Hartwig, currently in the publication process.

```
LOGIN_FAIL – PKG connection was established but login failed
for an unknown reason
BAD_USER_NAME – PKG connection was established but login
failed because of an unrecognized user name
BAD_USER_PW – PKG connection was established but login failed
because of an incorrect password
```

(2) The **num_req** function allows the caller to get a number of ORG-IDs. The number requested will be limited to MAX_R_OIDS. Upon successful completion this routine will return the value 1 and the requested ORG-IDs will be in the ids arra. No order can be assumed in the returned numbers. This routine blocks until a response is obtained.

```
int num_req ( req_num, ids, debug, eflag )
unsigned int req_num;  /* The number of ORG-IDs requested */
unsigned int *ids;    /* a pointer to sufficient space to store the
                         requested number of ORG-IDs */
  FILE *debug;        /* File descriptor for debug information. May
                         be NULL */
  int eflag;          /* Flag for encryption. 1 is on 0 is off */
  returns:
       BAD_ARGUMENT – One of the input arguments didn't make sense.
       Generally this is a pointer being NULL or a variable being
       out of reasonable range.
       PKG_SEND_FAILURE – PKG software send failed for the specified
       connection.
       PKG_WAIT_FAILURE – PKG software failed to get an expected
       response on the open connection.
       OVER_RL – Request refused – more hosts were requested than
       allowed during a single request
       OVER_DL – Request refused – more hosts were requested than
       allowed during a single day
       OVER_ML – Request refused – more hosts were requested than
       this OS is permitted to own
       REQ_FAIL – If the request failed for any one of many reasons
       other than the ones mentioned above
       num_ids – if successful, a positive number representing the
       number of OIDs granted is returned
```

(3) The **set_status** routine allows the OS program to set the status of ORGIDs assigned to it. Current status values are INACTIVE, ACTIVE, FREE. Ids are initially given a value of INACTIVE when assigned to an OS. The OS must then use this function to indicate those Ids that are issued b setting the status to ACTIVE. When the OS no longer has an use for selected Ids, the status may be set to FREE allowing the OIS to return them to the free list. Numbers less than 8 million can not be freed since these are preassigned to the various Oss. This routine blocks until a response is obtained.

```
  int set_status ( ids, num, status, debug, eflag )
unsigned int *ids;     /* a pointer to sufficient space to store
                          the requested number of ORG-IDs */
int num;          /* The number of entries in the ids array */
```

```
    FILE *debug;            /* File descriptor for debug information.
    May be NULL */
    int eflag;        /* Flag for encryption. 1 is on 0 is off */
    returns:
        BAD_ARGUMENT – One of the input arguments didn't make sense.
        Generally this is a pointer being NULL or a variable being
        out of reasonable range.
        PKG_SEND_FAILURE – PKG software send failed for the specified
        connection.
        PKG_WAIT_FAILURE – PKG software failed to get an expected
        response on the open connection.
        REQ_SUCCESS – for success
        REQ_FAIL – some problem was encountered and status value not
        changed for any of the ORG-ID numbers
```

(4) The **onwer_of** function allows the caller to determine the OS that is the owner of the specified ORGID. The result is returned in the string owner that is supplied b the caller. This routine blocks until a response is obtained.

```
  int owner_of ( id, owner, status, debug, eflag )
    unsigned int id;      /* An ORGID */
char *owner;          /* The name of the OS that owns this specified
ORGID. */
char status;          /* The status of the returned ORG-ID */
    FILE *debug;          /* File descriptor for debug information. May
    be NULL */
    int eflag;      /* Flag for encryption. 1 is on 0 is off */
    returns:
        BAD_ARGUMENT – One of the input arguments didn't make sense.
        Generally this is a pointer being NULL or a variable being out
        of reasonable range.
        PKG_SEND_FAILURE – PKG software send failed for the specified
        connection.
        PKG_WAIT_FAILURE – PKG software failed to get an expected
        response on the open connection.
        REQ_SUCCESS – for success
        REQ_FAIL – some problem encountered or couldn't find owner of
        this ORGID
```

(5) The **ois_do_logout** function terminates the connection to the OIS.

```
  int ois_do_logout (debug, eflag )
    FILE *debug;            /* File descriptor for debug information. May
    be NULL */
    int eflag;        /* Flag for encryption. 1 is on 0 is off */
    returns:
        BAD_ARGUMENT – One of the input arguments didn't make sense.
        Generally this is a pointer being NULL or a variable being out of
        reasonable range.
        PKG_SEND_FAILURE – PKG software send failed for the specified
        connection.
```

```
PKG_WAIT_FAILURE – PKG software failed to get an expected
response on the open connection.
LOGOUT_OK – logout succeeded and connection broken
```

I.4.3 Security Issues

A distributed computer system connected by the internet offers a number of security problems. It is impossible to make such a system perfectly safe, therefore any measures to protect such a system must be a compromise. The time, money and effort spent to protect the system must be balanced against what is being protected. To do this, potential vulnerabilities must be identified and then examined to determine how crucial each is to the overall system. Likewise, potential threats must be identified and the probability of them actually attacking the system be considered. In the case of a straightforward program like the OIS targets can be limited to

- The physical machine the OIS runs on
- The Operating System running on this machine
- The OIS program
- The Informix RDBMS (OIS DBS)
- The network (both between the OIS and OS and OIS1 and OIS2)
- Data in any of the above areas

The computers that host this OIS will be owned by the Army and reside on military establishments. As such, the machines can be assumed to be reasonably safe from physical attack.

The OIS software is composed of a C program that handles the networking connections with the various OS programs and an Informix RDBMS. A graphical user interface will allow the OIS manager to both manage and monitor the OIS. This software will run on the SOLARIS 2.6 operating system. Attacks on the OS or the Informix RDBMS are beyond the scope of this project.

The network is clearly the most obvious avenue of attack if, for no other reason, it's open to all the possible threats. Threats can literally be anyone, form hostile countries or terrorists, to curious high school students. Fortunately, the OIS operates on low value data. Although OIS is crucial to the entire ORG-ID concept, the numbers themselves are relatively unimportant. All the OIS program knows is which numbers are assigned to which OS and whether they are active or not. There is no association of numbers with units. Intercepting data between the OIS and OS is of limited usefulness.

The Primary threats to the OS are (1) denial of service, and (2) a "spoofer" program to falsely manipulate the OIS. If the OS can not obtain ORGIDs when needed, then the entire system fails.

Perhaps the easiest way to effect a denial of service is simply to flood the local network - denial of the network by overloading physical media.

In any client server system there are many opportunities for attack. Most attacks are focused at the endpoints and do not directly attack the encrypted messages passed back and forth. Here are some actions taken in the OIS and the API to implement a security policy. Some basic assumptions include that:

- The server machine is secure both in a software and physical sense.
- The network is non-secure
- The security of the Organization Server is not the problem of the OIS
- Data at the OIS is relatively non-valuable compared to data in the OS.

I.4.4 User Authentication

A need to authenticate clients is recognized as a concern. Security in the client/server model used in the OIS-OS system is provided by passwords and the limited number of legal users. Successful entry into the OIS requires that a previously established password be used. This password is then used to encrypt the entered password itself using a straightforward Data Encryption Standard (DES) algorithm[2]. During the initial login procedure the user name is sent in the clear and the encrypted password. At the OIS, this password is decrypted using the stored password and the are compared. A successful match constitutes a successful login, a session is established, and all further transactions are encrypted using the user's password.

Smartcard technology was considered but the overhead required for physical management of the smartcards was considered excessive. In addition, the possible access of the OIS program by allied nations may cause difficulties. Since it is expected the OS programs will connect to the OIS infrequently, it may be difficult to maintain synchronization with the smartcards.

I.4.5 Network Transaction Security

A DES algorithm is used to encrypt all transactions between the OIS and OS except for the initial login request. At this time it is not known if other procedures such as data randomization will be performed.

I.4.6 Data Storage Security

Data storage at the OIS site is the responsibility of the Informix RDMS. All Mission Critical information such as OID tables, user names and passwords will be stored by the RDMS. These data will then be replicated using the Informix replication scheme to the secondary OIS program.

References

1. Chamberlain, Samuel C., Web page: http://www.arl.mil/~wildman/PAPERS/tr2172.html,

Feb. 2000.
2. Young, Eric World Wide Web page :ftp://ftp.ps.uq.oz.au/pub/Crypto/DES/, 1998.

APPENDIX J

ACRONYM LIST

APPENDIX J - ACRONYM LIST

Acronym	Meaning
ABCS	Army Battle Command System
AFATDS	Army Field Artillery Tactical Data System
ASD(C3I)	Assistant Secretary of Defense for Command, Control, and Communications
AICDM	Army Integrated Core Data Model
AMC	Army Materiel Command
BRIL	Baseline Resource Items List
BUOF	Building Units on the Fly
CADM	Core Architecture Data Model
CCB	Configuration Control Board
CDAd	Component Data Administrator
COE	Common Operating Environment
CSSCS	Combat Service Support Control System
C2	Command and Control
C2CDM	C2 Core Data Model
C2CRM	C2 Core Repository Model
C4ISR	Command, Control Communications, Computer, Intelligence, Surveillance and Reconnaissance
DBLS	Distribution Based Logistics System
DBMS	Database Management System
DDA	Defense Data Architecture
DDL	Data Definition Language
DDDS	Defense Data Dictionary System
DIA	Defense Intelligence Agency
DII	Defense Information Infrastructure
DISA	Defense Information Systems Agency
DLA	Defense Logistics Agency
DLIS	Defense Logistics Information Service
DM	Data Model
DML	Data Manipulation Language
DODAAC	Department of Defense Activity Address Code or Address Accounting Code
DOO	Default Operational Organization
DUI	Data Use Identifier
EI	Enterprise Identifier
EK	Enterprise Key

Acronym	**Meaning**
ERD	Entity Relationship Diagram
FDAd	Functional Data Administrator
FIPS	Federal Information Processing Standard
FMS	Force Management System
FSC	Federal Supply Class
FSG	Federal Supply Group
GIG	Global Information Grid
IMMC	Integrated Materiel Management Center
IP	Internet Protocol
JCDB	Joint Common Database
JTF	Joint Task Force
JVMF	Joint Variable Message Format
LAISO	Lead AMC Integration Support Office
LC2IEDM	Land C2 Information Exchange Data Model
LIDB	Logistics Integrated Data Base
LOGSA	Logistics Support Agency
MEU	Marine Expeditionary Unit
MTOE	Modified Table of Organization and Equipment
NCB	National Codification Bureau
NIIN	National Item Identification Number
NSN	National Stock Number
ODCSLOG	Office of the Deputy Chief of Staff for Logistics
ODCSOPS	Office of the Deputy Chief of Staff for Operations
OOB	Order of Battle
OPORD	Operations Order
RDBMS	Relational Database Management System
SHADE	Shared Data Environment
SMD	Ship Manning Document
SQL	Structured Query Language
SSN	Social Security Number
STAMIS	Standard Army Information Systems
TI	Tactical Internet
TIDP	Technical Interface Design Plan
TF	Task Force

Acronym	**Meaning**
TPFDD	Time Phased Force and Deployment Data
TPIO	TRADOC Program Integration Office
TRADOC	Training and Doctrine Command
UIC	Unit Identification Code
UIR	Uniform Resource Identifier
UMD	Unit Manning Document
URN	Unit Reference Number
USD(AT&L)	Under Secretary of Defense (Acquisition Technology & Logistics
USAFMSA	United States Army Force Management Support Agency
VMF	Variable Message Format
XML	Extensible Mark-up Language
WIN-T	Warfighter Information Network - Tactical

APPENDIX K

IMPLEMENTATION PLAN FOR ENTERPRISE IDENTIFIERS (DRAFT)

1.0 EXECUTIVE SUMMARY

"Our commitment to meeting these challenges compels comprehensive transformation of The Army. To this end, we will begin immediately to transition the entire Army into a force that is strategically responsive and dominant at every point on the spectrum of operations. We will jumpstart the process by investing in today's off-the-shelf technology to stimulate the development of doctrine, organizational design, and leader training even as we begin a search for new technologies for the objective force. Doing so will extend our technological overmatch."

General Eric K. Shinseki

As the words of General Shinseki indicate, the new Army must be "responsive and dominant at every point in the spectrum of operations." It is highly unlikely that this will be accomplished unless we develop the capability to rapidly designate, marshal, deploy, and employ joint, tailored military forces designed to meet specific needs and requirements in response to a wide range of possible worldwide contingencies.

Therefore, the capability must exist for field commanders to rapidly restructure the forces under their command to respond to changing battlefield situations. To realize General Shinseki's vision a commander must have the capability of accessing the three principal force package data elements, namely, the organization structure, the personnel resources, and the appropriate materiel complement, with confidence in their accuracy and reliability. In spite of the large number of information systems, the current situation is such that commanders are not able to obtain timely delivery of these capabilities. Untimely delivery of critical data is largely the result of the following deficiencies: (1) lack of a single, unifying concept for information management, and (2) lack of unambiguous sources of standard data.

Proposed Solution

1. Identify the authoritative data sources within the Army and assign to them ownership and responsibility for their respective data sets. These data sets are the ones needed to support the warfighter throughout the entire spectrum of combat operations (e.g., Army Battle Command System (ABCS), Global Combat Support System – Army (GCSS-A), etc.).

2. Mandate that all authoritative data sources adopt an identifying convention or enterprise identifier (EID) scheme using 64-bit keys, for locating and managing their respective data sets. The EID scheme will allow all objects across the combat, combat support, and combat service support domains to be uniquely identified.

3. Implement the 64-bit EID allocation scheme, for use by all authoritative data sources, by obtaining the first half of the EID from a 32-bit key seed-server (initially, the ORG-ID Server),

and allowing the other 32-bit half of the EID to be generated and managed by the designated authoritative data sources via servers created for this purpose (i.e., EID Servers).

4. Provide policies and procedures for establishing and maintaining EID servers to ensure that EIDs are allocated and tracked in a uniform and rigorous manner.

5. Provide policies and procedures to ensure that all information systems in support of the warfighter can use the reference data sets resident in the authoritative data sources by using the EIDs assigned to them.

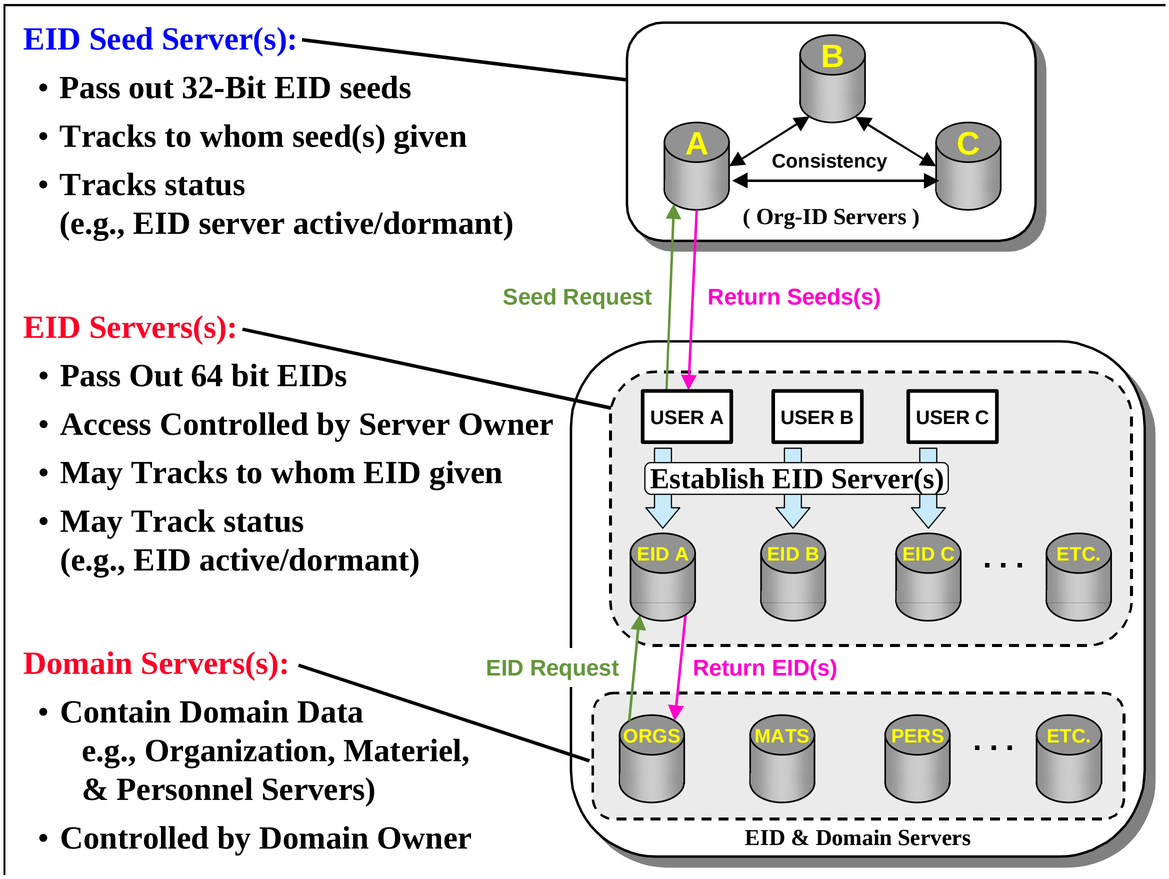


Figure A. EID Seed Server Provides Seeds To Users Who Establish EID Servers to Facilitate Populating Their Domain Servers

2.0 EID IMPLEMENTATION PLAN

I enthusiastically support the implementation of the ORG-ID concept. The ORG-ID is the key in establishing and maintaining the Joint Common Database. With force structure information directly feeding Army tactical systems the battlefield commander, and his staff, will see timely Situation Awareness, and have the ability to more proactively execute missions. This concept touches all aspects of digitization and reduces the manual burden we place upon our soldiers. Reduced costs in manpower, time, dollars, and bandwidth are significant as information is better managed, distributed and re-used. Our Joint and Coalition digital interoperability initiatives are also reliant upon ORG-ID, and the Joint Common Database. Logisticians will know the current status of battalions and get the right material, to the right place, at the right time. The G-1 will have the current personnel status of a division at the push of a button, taking the guesswork out of personnel replacement operations. With fewer people inputting information, and then globally sharing this data, I see some real savings in costs, time, and equipment.

LTG William M. Steele

2.1 Introduction

At the database level, that is, at the actual physically implemented level, all automated information systems (AISs) must be able to uniquely identify individual pieces of data, whether they be objects or records. For example, relational databases represent their contents in the form of records. Every record must be addressable for the AIS to fulfill its role with regard to the user. To that effect, databases use a special type of field called a "key". One characteristic of database keys is that they have to be unique within the table in which they are defined, because tables are the basis for queries that create, retrieve, update, or delete (CRUD) the actual information contained in the records. If two or more records have the same key within the same table, CRUD operations would not be able to resolve which of the two or more records is the one to act on.

If an AIS operates independent from any other AIS within the enterprise, then there is no need to worry about its key management scheme, other than to satisfy the minimum requirement of uniqueness within the database tables. However, this scenario is becoming less and less likely as we realize the advantages of pulling together resources and leveraging disparate, pre-existing data, thereby saving money and time.

When multiple AISs are required to exchange data one often confronts the issue of data interoperability. At its most basic level data interoperability can be understood as the ability of system A to re-use part or all of the data contained in system B without any need to manipulate or change the data prior to its consumption. This means that semantically[80] the objects in system A and system B are identical, e.g., TANK LENGTH DIMENSION in system A means the same as in system B. Note that implementers are free to name the physical elements differently within each system, as long as the element is defined in the same way. In other words, system A could label TANK LENGTH DIMENSION as TNK_LNGTH and system B could label TANK LENGTH DIMENSION as TNKLDIM within their respective tables, but both systems would share the same data type, e.g., FIX (13,2), with unit of measure METER. Under these circumstances, system A could pass to system B all its records containing TANK LENGTH DIMENSION data without any need to transform them before re-use.

If some organization within the enterprise is designated as the authoritative source for TANK LENGTH DIMENSION data, then it would be very useful if a special table of TANK LENGTH DIMENSION data were created and disseminated ahead of time. Instead of replicating the data collected and maintained by this organization it could be re-used by other AISs within the enterprise simply by referencing the records in the special table via the keys that have been allocated to them. To do this the keys now need to be unique not only within the table in system A, but unique across all tables in all participating AISs. This is one feature that the adoption of globally unique Enterprise Identifiers (EIDs) enables the participants to accomplish.

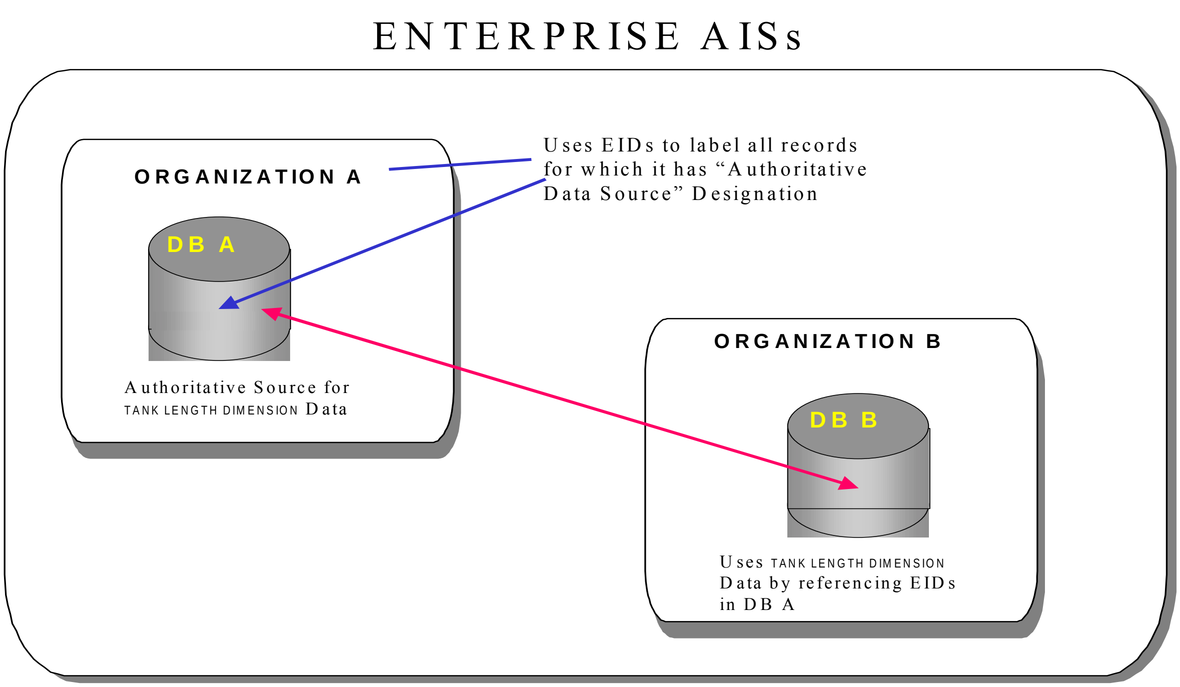


Figure 2-1. Adoption of EIDs within the Enterprise enables Re-Use by all Participants of Authoritative Source Data Sets via Reference to EIDs

Figure 2-1 above depicts graphically the way in which AISs that adopt the EID key scheme within the enterprise can accomplish re-use of authoritative source data sets. System B needs only to know the EIDs assigned to the TANK LENGTH DIMENSION data by organization A within Database A (DB A) to build a report with data contained in Database B.

---

[80] Semantics: the study of meanings. In other words, "the meaning of;" (http://www.m-w.com/cgi-bin/dictionary).

The advantage of this approach is that data can now be "entered-once" by the organization that has been given "Authoritative Data Source" designation, but it can be "re-used many times" by all the participants, since all that is required is the EIDs pointing to that data. Organization A can perform CRUD operations on its data set, and users will always have the most recent information, since at report time they all access the most up-to-date records from the authoritative source.

Data sets can be created by organizations which have "Authoritative Data Source" designation. These data sets can be pre-loaded into the AISs that support the warfighter allowing messages to be exchanged among AISs that contain only the EIDs. This is possible because the common "look-up" tables are guaranteed to contain unambiguously data via identical EIDs.. This saves bandwidth and minimizes the affects of communication disconnects among the enterprise participants.

2.2 Implementation Steps

As explained in Section 2.1, the use of EIDs presupposes a series of steps to fully deliver all its benefits. For the Army enterprise, it is recommended the Army CIO proceed as follows:

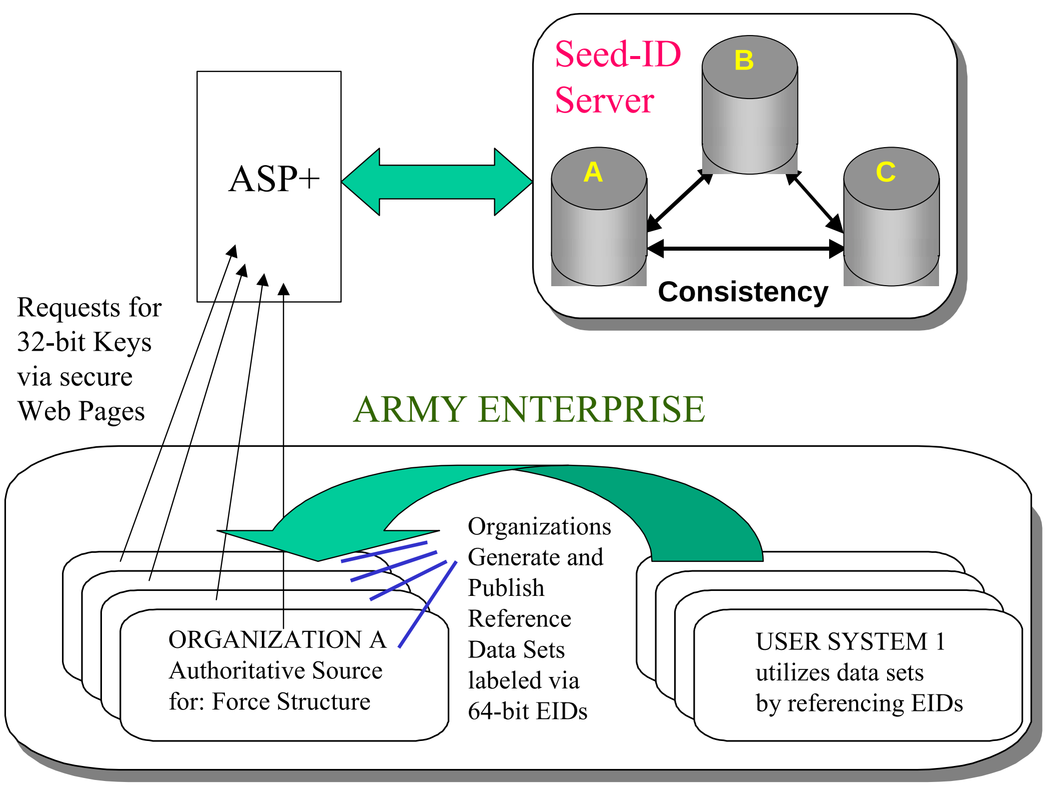


Figure 2-2. Implementation Concept for Use of EIDs within the Army Enterprise

1. Identify the authoritative data sources within the Army and assign to them ownership and responsibility for their respective data sets. These data sets are the ones required to support the

warfighter throughout the entire spectrum of combat operations. For example, LOGSA which currently maintains a repository of Army Logistics data in the Logistics Integrated Database (LIDB), could be designated as the "Authoritative Data Source" for all the materiel classes with which Army units are equipped. The U.S. Army Force Management Support Agency (USAFMSA) is the "Authoritative Data Source" for all the Army's force structure data and the ORD for their new Force Management System includes the requirement to build an organization server for this purpose. Army Human Resources (PERSCOM) could be designated as the "Authoritative Data Source" for all the Army's personnel data.

2 Establish policies and procedures for establishing and maintaining EID servers to ensure that EIDs are allocated and tracked in a uniform and rigorous manner. This should include procedures for the disestablishment of EID servers, perhaps with an option to "return" to the EID Seed server those 32-bit keys that they do not plan to use, as well as procedures for re-issuance of those keys once it has been established that no system within the enterprise has created 64-bit EIDs with them.

3 Establish procedures to access the 32-bit key EID Seed server so that authorized users can establish an EID server to generate 64-bit EIDs to label their records. At a minimum the 32-bit key EID Seed server must be password protected, and accessible via Web-browsers. Since the 32-bit key EID Seed server can distribute up to 4.3 billion different 32-bit keys, the CIO may impose restrictions on the number of keys that any given organization can request on a regular basis. Additionally, a set of 32-bit keys in the EID Seed server may be reserved for special use, e.g., to allow an organization who may not come on line until later to have a sufficient number of 32-bit keys once it begins its implementation. The overall approach is depicted in Figure 2-2.

3.0 CONCLUSIONS

The Army is one of the largest users of automation for collecting, maintaining and using data to support its activities throughout the whole spectrum of combat operations. At present, in spite of the numbers of AISs that the Army has implemented, information cannot be readily and accurately retrieved from these systems. In addition, vast numbers of these systems replicate data out of a few sources with varying degrees of fidelity thereby exacerbating the problem.

Recent developments in database and web technologies have made it possible to envision a scenario in which data can be "entered-once" and then be "reused-many" times with minimal danger of ambiguity.

One easy and robust way for making this vision a reality appears to be the adoption of a scheme to label "authoritative data set" records in a globally unique manner within the whole Army enterprise. The approach is based on the acceptance by the participating organizations within the Army enterprise of the use of 64-bit keys that are guaranteed to be globally unique by virtue of being generated out of a 32-bit key that an EID Seed Server distributes to them. Once each system acquires such a 32-bit key it can then, by concatenation of a second 32-bit number, manage up to 4.3 billion records. The 64-bit keys (EIDs) generated in this manner make it highly unlikely that two distinct records can be given the same identifier.

If the entities respectively in charge of the Organization, Materiel and Person domains implement such an approach, the Army could begin to create applications that provide the capability for Building Units on the Fly (BUOF), as well as faster generation (e.g., 72-hr) of the Time Phased Force and Deployment Data (TPFDD).

The introduction of EIDs may also provide a smooth and efficient migration path to achieving data interoperability. Implementation of enterprise identifiers can be accomplished in a way that will minimize any cost impact on legacy systems–e.g., rewriting of code. If an EID is placed as an alternate key in a database, no code or SQL has to be rewritten. To create EIDs, all that is needed is a procedure included in the DBMS INSERT routine. This is not a very intrusive operation. Organizations which introduce EIDs into their database tables as primary keys may want to maintain their UICs and/or DODAACs as alternate keys. If this approach is adopted, the "business keys" (and many others) will be around as long as people want them. However, it is hoped that if enterprise identifiers are provided, of which Org-EIDs are a variant, software engineers will exploit the many benefits of using them. Eventually, UICs and DODAACs might naturally disappear. But in the short term, there is no change to the current (large) set of choices to identify a unit across Army, joint, and coalition boundaries.

Ultimately, the Army vision is a set of integrated servers with up-to-date default information on (initially) FORCE STRUCTURE, MATERIEL, and PERSONNEL. This information should be:

(a) maintained by the persons (authoritative sources) who do it for a living, and
(b) easily loaded (or eventually downloaded) into tactical systems.

Enterprise identifier management is the chief means of exchanging data in a shared data environment. The benefits of enterprise identifiers are obvious. The problem, however, is the architectural arrangement of servers (i.e., what information should be kept at which servers; what information to release from a given server; what ontology or way of classifying and naming information is most efficient and effective when querying the servers; etc.). Clearly, there are numerous issues that must be addressed before data interoperability is a reality across mission areas.

The current situation with stovepipe databases underlines the concern of how much it will cost the Army over the next twenty years to delay evolving to EIDs. It is probably fair to say that this cost will far exceed any costs of moving to EIDs. The issue is simple. The Army cannot keep kludging together systems with tenuous band-aids. Short-term costs may appear attractive, but the accumulated costs are immense. It is expected that the Army, over the next 20 years, will spend a hundred times the conversion cost if it wants to integrate its systems without moving to a common convention for record identification and management. In fact, there appears to be no affordable way to integrate Army (e.g., ABCS, GCSS-A, etc.), joint, and coalition systems without a common format for record identification and management. Arguably, the adoption of a common 64 bit surrogate key structure is the most fundamental prerequisite [not to mention that it has to be terse so that it works over SINCGARS]. Realistically, all that is meant thereby is the standardization of ONE FIELD, not hundreds of tables, objects, or a whole GIG architecture. If the Army can't accomplish this, it is doubtful that it will succeed when attempting the "hard stuff," like integrated planning, targeting, supply-chain, etc.

Endorsement by the Army CIO of the implementation plan discussed in the previous sections is required. The implementation plan is consistent with Army Transformation Initiatives and supports Gen. Shinseki's vision for an Army that is responsive and dominant at every point on the spectrum of operations.